Titre: Title:	Étude, conception et validation d'une technique efficace d'élimination en temps-réel des ECG dans les EMGdi
Auteur: Author:	Bassam Rhou
Date:	2010
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Rhou, B. (2010). Étude, conception et validation d'une technique efficace d'élimination en temps-réel des ECG dans les EMGdi [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/254/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/254/
Directeurs de recherche: Advisors:	Mohamad Sawan
Programme: Program:	Génie électrique

UNIVERSITÉ DE MONTRÉAL

ÉTUDE, CONCEPTION ET VALIDATION D'UNE TECHNIQUE EFFICACE D'ÉLIMINATION EN TEMPS-RÉEL DES ECG DANS LES EMGdi

BASSAM RHOU

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

MARS 2010

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

ÉTUDE, CONCEPTION ET VALIDATION D'UNE TECHNIQUE EFFICACE D'ÉLIMINATION EN TEMPS-RÉEL DES ECG DANS LES EMGdi

Présenté par : RHOU Bassam

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées

a été dûment accepté par le jury d'examen constitué de :

M. SAVARD Pierre, PhD, président

M. SAWAN Mohamad, PhD, membre et directeur de recherche

M. ZHOU Guchuan, Doct., membre

DÉDICACE

À mes parents…

REMERCIEMENTS

Tout d'abord, je tiens à remercier mon directeur de recherche, Pr. Mohamad Sawan, pour avoir encadré mon travail de recherche tout au long de ma Maîtrise et pour m'avoir donné la chance de travailler sur un projet aussi intéressant tout en me soutenant financièrement.

Je tiens à remercier les membres de l'équipe Polystim pour leur esprit de camaraderie et leur disponibilité ainsi que les techniciens du département de génie électrique.

Je désire également remercier mes parents Najat et Alhassan et ma sœur Chaymaa pour leurs encouragements et leur soutien tout au long de mes études de Maîtrise malgré la distance qui nous sépare.

Finalement, je remercie les membres du jury qui ont accepté d'évaluer la qualité de mon travail.

RÉSUMÉ

Les EMG sont les signaux électriques caractérisant l'activité neuromusculaire d'un muscle donné. Ces signaux sont utilisés pour diverses applications médicales comme le diagnostic des anomalies dans les fonctions musculaires et le contrôle des prothèses myoélectriques.

Ces dernières années, les signaux électromyogrammes diaphragmatiques (EMGdi) ont été de plus en plus utilisés et plusieurs recherches ont été faites pour les exploiter dans des applications médicales classiques comme le diagnostique des anomalies respiratoires ou des applications novatrices comme le contrôle de la respiration artificielle.

La principale difficulté pour exploiter les EMG en général et les EMGdi en particulier est restée l'élimination de la contamination par les ECG. En effet, l'EMG acquis via une électrode représente la somme de tous les potentiels d'action des unités motrices d'un muscle donné dans la zone de détection de cette électrode. Cependant, le signal électrique résultant de l'activité cardiaque est si puissant qu'il peut être détecté par ces électrodes. Ceci implique que le signal EMG détecté est toujours contaminé par les artéfacts cardiaques, ce qui implique une erreur dans la valeur du RMS et rend l'information électrique obtenue sur un muscle donné moins précise. Un filtrage simple ne peut pas éliminer cette contamination vu que les domaines spectraux des EMG et des ECG ne sont pas distincts et une technique de filtrage complexe doit être mise en place afin d'obtenir des résultats facilement exploitables. Cependant, deux autres difficultés entrent en jeu lorsqu'il s'agit d'applications de contrôle de prothèses ou de contrôle de la respiration artificielle. La première difficulté est l'obtention de signaux EMG directement exploitables en temps réel car, dans ce cas, l'élimination des ECG doit aussi être faite en temps

réel. La seconde difficulté est la détection automatique de l'activité musculaire à partir de ces signaux, ce qui implique l'utilisation d'un algorithme pour détecter les blocs d'EMG.

Afin de trouver des solutions efficaces aux difficultés présentées précédemment, le présent projet propose une solution qui a été développée en trois étapes.

La première étape consiste en une comparaison des techniques d'élimination des ECG dans les EMG déjà proposées dans le passé afin de choisir la technique la plus performante et la plus convenable pour des applications en temps réel. Les techniques proposées dans la littérature seront d'abord comparées afin de localiser les techniques pouvant opérer en temps réel et d'une manière automatique. Par la suite, les techniques localisées seront adaptées aux EMGdi, puis comparées à l'aide d'un logiciel qu'on a réalisé en langage LabView implémentant ces techniques afin de choisir la technique la plus efficace pour l'élimination des ECG dans les EMGdi. Une comparaison entre les méthodes de détection de l'activité musculaire prendra également place dans cette partie afin de présenter les avantages et les inconvénients des méthodes de détection existantes.

Dans un second lieu, la méthode choisie dans la première étape sera validée matériellement. Cette validation a été faite sur un système basé sur deux microcontrôleurs. Par la suite, une architecture implémentant la technique choisie sera proposée et implémentée en langage VHDL.

Finalement, une nouvelle méthode basée sur la méthode choisie et implémentée sera proposée et ses performances seront évaluées par rapports aux méthodes actuelles. Une méthode de

détection automatique de l'activité musculaire combinée à la technique choisie sera également proposée.

Le logiciel Labview développé, le système basé sur des microcontrôleurs, l'implémentation en VHDL ainsi que la technique d'élimination des ECG améliorée et de la détection de l'activité musculaire sont fonctionnels.

ABSTRACT

EMG are electrical signals characterizing the neuromuscular activity of a given muscle. These signals are used for various medical applications such as abnormal muscle function diagnostic and myoelectric prosthesis control.

Recently, the diaphragmatic electromyogram signals (EMGdi) were used increasingly and several research dealed with the use of EMGdi in classical medical applications such as diagnostic of respiratory diseases or innovative applications such as control of mechanical ventilation.

The main difficulty for EMG using is ECG cancellation. Indeed, the EMG obtained using an electrode is the sum of all motor unit action potentials of a given muscle in the detection area of this electrode. However, the cardiac signal is so powerful that it can be detected by these electrodes. Thus, detected EMG is always contaminated by cardiac artefacts implying a RMS error and making the electrical information obtained from a given muscle less accurate. A simple filter can not eliminate this contamination because the spectra of EMG and ECG overlap. Thus, there is a need for a complex filter to obtain easily exploitable results. However, two more difficulties appear when the EMG signal is used for prosthesis control or for mechanical ventillation. The first difficulty is obtaining EMG signals which can be used directly in real-time because, in this case, ECG cancellation should be done in real-time. The second difficulty is automatic detection of muscular activity using these signals because this operation needs the use of an complex algorithm to detect EMG activation and deactivation.

In order to find efficient solutions for the previous difficulties, this project present a new solution that was developed in three steps.

The first step is a comparison of techniques for ECG cancellation in EMG signals already proposed. This comparison is made to choose the most powerful and suitable technique for real-time applications. The existing techniques will be compared to identify the technique that can operate in real-time and automatically. Thereafter, the identified techniques will be adapted for EMGdi and compared using a LabView software to choose the most efficient method. A comparison of muscular activity detection methods will also be made in order to present the advantages and disadvantages of each method.

In a second part, the method that was chosen in the previous step will be validated in a hardware system. This validation will be done using a system based on two microcontrollers. Thereafter, an architecture implementing the chosen method will be proposed and implemented in VHDL.

Finally, a new method based on the chosen technique will be proposed and its performance will be evaluated compared to current methods. A global method combining ECG cancellation and muscular activity detection will also be proposed.

The LabView software, the system based on microcontrollers, the VHDL implementation and the global method are working correctly.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ARV	Average Rectified Value
BW	Butterworth
CAN	Convertisseur Analogique/Numérique
CISC	Complex Instruction Set Computer
CWT	Continuous Wavelet Transform
ECG	Électrocardiogramme
EMG	Électromyogramme
EMGdi	Électromyogramme diaphragmatique
ICA	Independent Component Analysis
IHM	Interface Homme/Machine
Ko	Kilooctet
MEMS	MicroElectroMechanical System (Système microélectromécanique)
MH	Méthode Hybride
MMF	Moyenne Mobile Finie
MUAP	Motor Unit Action Potential
NAVA	Neurally Adjusted Ventilatory Assist
NTH	Nouvelle Technique Hybride

Pdi	Pression transdiaphragmatique
RISC	Reduced Instruction Set Computer
RMS	Root Mean Square (valeur efficace)
RS-232	Recommended Standard 232
SC	Spike-Clipping algorithm
SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface
TTL	Transistor-Transistor Logic USART
VHDL	Very High Speed Integrated Circuit Hardware Description language
VI	Virtual Instrument

LISTE DES ANNEXES

INTRODUCTION

MOTIVATIONS

L'électromyogramme (EMG) est un signal biologique représentant la manifestation électrique de l'activité neuromusculaire d'un muscle contracté du corps. Cette manifestation électrique permet de fournir des informations importantes sur l'état d'un muscle du corps. Ces informations, une fois traitées, peuvent être utilisées par un médecin ou un système afin de donner le diagnostic ou la réaction la plus appropriée. Durant ces trente dernières années, l'EMG a été de plus en plus un sujet d'intérêt pour les chercheurs grâce aux développements technologiques qui ont permis l'acquisition et le traitement de ce signal beaucoup plus facilement que dans le passé.

Dernièrement, plusieurs techniques d'acquisition de l'EMG diaphragmatique (EMGdi) ont été développées, ce qui a élargit le champ d'application de ces signaux pour inclure le contrôle automatique de la ventilation artificielle. En effet, l'utilisation des EMGdi pour contrôler automatiquement la respiration artificielle est une alternative efficace aux ventilateurs mécaniques classiques qui fournissent une pression constante aux poumons pendant la période inspiratoire quelque soit l'effort respiratoire du patient. Les ventilateurs mécaniques classiques peuvent, à cause de leur fonctionnement, causer des barotraumatismes lorsque l'effort respiratoire fourni par le patient ajouté à celui fourni par le ventilateur dépasse un certain seuil [Loring et Malhotra, 2007]. Ce problème peut être évité si l'effort respiratoire fourni par le ventilateur varie dépendamment de l'effort fourni par le diaphragme du patient. Ainsi, une méthode de ventilation mécanique qui consiste à fournir un effort proportionnel à l'EMGdi a

été envisagée [Sinderby *et al.*, 2007]. Cette méthode, baptisée NAVA (*Neurally Adjusted Ventilatory Assist*), consiste à fournir une pression positive dans les voies respiratoires proportionnelle à l'activité électrique du diaphragme. Lorsque l'activité électrique du diaphragme est terminée, la pression n'est plus délivrée.

L'EMGdi, ainsi que sa valeur RMS, sont également très utiles pour le diagnostique de l'insuffisance respiratoire aiguë. En effet, l'EMGdi permet d'évaluer l'activité électrique des centres respiratoires. La pression transdiaphragmatique, notée par la suite Pdi, représente la différence entre les pressions œsophagienne et gastrique. Cette pression, une fois jumelée à l'EMGdi, permet de donner une caractérisation du couplage électromécanique du diaphragme et permet ainsi de trouver plus efficacement la source provoquant l'insuffisance respiratoire chez un patient.

Afin d'avoir un système permettant de fournir des données pouvant être utilisées pour caractériser le couplage électromécanique du diaphragme, un cathéter œsophagien sans fil permettant l'acquisition conjointe et en temps réel des signaux EMGdi et Pdi a été réalisé au laboratoire Polystim de l'École Polytechnique de Montréal [Désilets *et al.*, 2006]. Ce système est destiné à court terme au diagnostic de l'insuffisance respiratoire aiguë, et à long terme au contrôle automatique de la respiration artificielle. L'acquisition de l'EMGdi et de la Pdi se fait par le biais d'un cathéter œsophagien composé de cinq électrodes anneaux pour l'acquisition des EMGdi et de deux capteurs de pression pour l'acquisition de la Pdi. Ce cathéter est connecté à une carte qui permet l'amplification et le traitement des signaux acquis, ainsi que la communication avec une interface homme/machine via une liaison Bluetooth. L'utilisation de ce cathéter permet d'inclure, en un seul système hybride, l'acquisition des EMGdi et de la Pdi et

donne une bonne alternative à l'utilisation d'autres moyens d'acquisition comme les ballonnets et les aiguilles électrodes qui peuvent être invasifs et encombrants [Dido, 2002]. L'acquisition des signaux respiratoires est actuellement fonctionnelle sur ce système. Cependant, les signaux EMGdi acquis sont contaminés par le bruit cardiaque. Afin d'avoir des signaux EMGdi exploitables pour le diagnostic des disfonctionnements neurologiques respiratoires ou pour l'utilisation future de ce système comme contrôleur de la ventilation artificielle, cette contamination doit être éliminée. Une fonction consistant à détecter les déclenchements et les arrêts de l'activité du diagramme devrait également être implémentée dans le cadre de l'adaptation de ce système d'acquisition à la fonctionnalité du contrôle de la respiration artificielle. Les signaux EMGdi, n'ayant pas une amplitude ni une fréquence fixes, font en sorte que la détection de l'activité musculaire du diaphragme ne soit pas une simple opération et une méthode complexe s'impose.

Afin de contrôler un ventilateur mécanique en se servant de l'EMGdi pour fournir un effort respiratoire approprié, les EMGdi doivent avoir un rapport signal sur bruit élevé et l'acquisition devrait se faire en temps réel et d'une manière automatique. Cependant, les méthodes actuelles d'acquisition des EMGdi ne permettent que l'acquisition d'un EMGdi contaminé par les ECG et parfois par des artéfacts de mouvement. La contamination pas les ECG n'est pas facile à éliminer par un filtre classique car les EMG et les ECG n'ont pas des domaines spectraux distincts. Une technique plus complexe opérant d'une manière automatique et en temps réel s'impose.

Plusieurs méthodes d'élimination des ECG dans les EMG ont été proposées dans le passé. Cependant, ces méthodes proposées nécessitent pour la plupart un délai de traitement de

données important ou nécessitent un traitement après l'acquisition du signal EMG en entier, ce qui les rend inopérables sur un système temps-réel. Ajouté à cela, certaines de ces techniques ne peuvent pas opérer d'une manière autonome dans un système temps réel et nécessitent l'intervention d'un opérateur. Nous aborderons ces méthodes d'une façon détaillée dans le prochain chapitre de ce mémoire.

Objectifs

La contamination des signaux EMGdi par les signaux ECG expliquée précédemment limite l'utilisation des EMGdi pour des fins de diagnostic ou de contrôle de la respiration artificielle. Cette contamination complique également la détection de l'activité du diaphragme basée sur l'EMGdi. Le présent projet a pour but de proposer une technique efficace d'élimination des ECG dans les EMGdi ainsi qu'une technique de détection de l'activité du diaphragme en temps réel et d'une manière automatique. Par conséquent, deux aspects seront abordés dans ce projet :

Le premier aspect concerne l'élimination des ECG de l'EMGdi. Tout d'abord, on présentera une étude comparative des méthodes actuelles d'élimination des ECG dans les EMG en se basant sur les publications scientifiques concernant ces techniques. Les techniques pouvant être utilisées en temps réel et qui peuvent opérer d'une manière autonome tout en donnant des résultats satisfaisants seront par la suite adaptées pour convenir aux signaux EMGdi obtenus par le système d'acquisition décrit précédemment [Désilets et al., 2006], puis implémentées en langage LabView afin de choisir la méthode la plus efficace. La méthode la plus efficace sera par la suite validée matériellement sur une carte à microcontrôleurs et une architecture VHDL implémentant cette méthode sera proposée. En dernier lieu, une nouvelle méthode basée sur la méthode choisie sera proposée et implémentée en langage LabView.

Le deuxième aspect concerne la détection de l'activité musculaire du diaphragme. Une étude comparative des techniques de détection de l'activité musculaire sera faite et une nouvelle technique de détection en temps réel et d'une manière autonome de l'activité musculaire du diaphragme sera proposée. La technique proposée englobe la technique d'élimination des ECG dans les EMGdi, présentée dans le premier aspect, ainsi qu'un algorithme à moyenne mobile finie.

Nous nous concentrons dans ce mémoire sur l'aspect électronique du projet. Nous utiliserons un signal EMGdi d'une durée de 36 secondes pour montrer la précision de l'algorithme de traitement proposé et le bon fonctionnement du système implémenté. Cependant, la validation pour un but de certification ne sera abordée dans ce mémoire étant donné que cette dernière implique d'avoir une large base de données de signaux EMGdi.

Organisation du mémoire

Le premier chapitre expose les différentes méthodes connues de l'élimination des ECG dans les EMG et de détection de l'activité musculaire à partir du signal EMG d'un muscle donné.

Le second chapitre est consacré à une étude comparative des méthodes d'élimination des ECG dans les EMG et des méthodes de détection de l'activité musculaire à partir du signal EMG exposées dans le premier chapitre.

Le troisième chapitre propose une validation matérielle par un système à microcontrôleurs et par une architecture VHDL de la technique d'élimination des ECG dans les EMG choisie dans le second chapitre. Ce chapitre présente également une nouvelle méthode d'élimination, en temps réel et d'une manière automatique, des ECG dans les EMGdi ainsi qu'une nouvelle

technique de détection, en temps réel et d'une manière automatique, de l'activité du diaphragme. Ces deux techniques seront combinées en une seule méthode permettant l'élimination de la contamination ECG dans les EMG et la détection simultanée de l'activité musculaire.

Les résultats obtenus par les systèmes proposés dans le troisième chapitre seront présentés dans un quatrième chapitre. Une conclusion et des recommandations seront exposées dans un cinquième chapitre.

CHAPITRE 1 MÉTHODE D'ÉLIMINATION DES ECG DANS LES EMG ET DE DÉTECTION DE L'ACTIVITÉ MUSCULAIRE À PARTIR DU SIGNAL EMG

1.1 Problématique

Le présent travail s'inscrit dans le cadre d'un projet de plus grande envergure visant à réaliser un système autonome d'acquisition simultanée des signaux EMGdi[1] et des signaux Pdi (figure 1.1) pour des fins de diagnostic et de contrôle automatique de la respiration artificielle. Ce système est composé d'un cathéter œsophagien, sur lequel sont fixés deux capteurs MEMS pour l'acquisition de la Pdi et cinq électrodes anneaux équidistants pour l'acquisition de l'EMGdi et d'une carte électronique externe.

Le fonctionnement de ce système est comme suit : les signaux EMGdi et Pdi sont acquis via un cathéter inséré dans l'œsophage. Ces signaux sont par la suite véhiculés jusqu'à la carte électronique externe qui fera les traitements nécessaires (amplification, échantillonnage, gestion et contrôle de la réception et de l'envoi des données), puis envoyés sous forme numérique via une liaison sans fil Bluetooth à une interface utilisateur pour être affichés et sauvegardés dans un fichier texte.

[1] Une définition détaillée des signaux EMG est donnée dans l'annexe A.

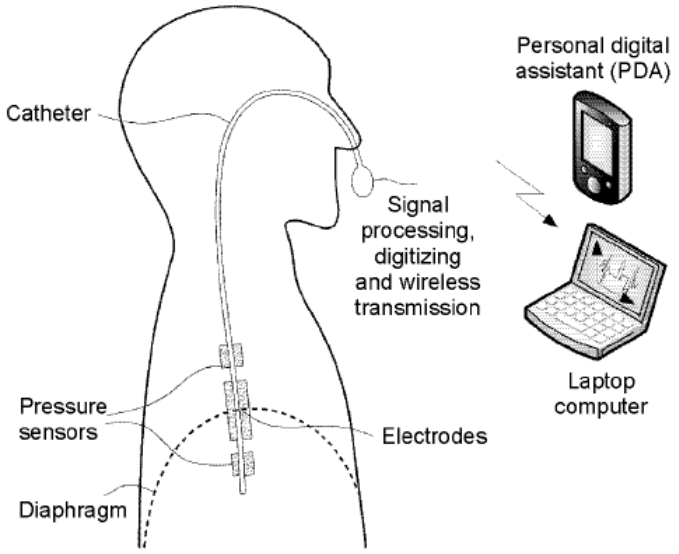


Figure 1.1 : Système d'acquisition des signaux respiratoires [Désilets et al., 2006]

Comme dans tous les systèmes d'acquisition des EMG, les signaux EMGdi acquis sont contaminés par les signaux cardiaques ECG. La figure 1.2 montre un exemple de signaux EMGdi acquis via ce système. Cette contamination peut mener à des mauvaises interprétations sur le muscle étudié et rend difficile la détection automatique de l'activité de ce muscle.

L'élimination efficace des ECG contaminant le signal EMGdi présenté dans la figure 1.2 n'est pas chose simple. En effet, le filtrage classique n'élimine pas tous les signaux ECG contaminant le signal EMGdi car les bandes spectrales des signaux EMG et ECG ne sont pas distinctes (cf. figure 1.3). Ainsi, l'utilisation de techniques de filtrage exploitant d'autres points de différences entre les EMG et les ECG est nécessaire pour obtenir des résultats plus satisfaisants.

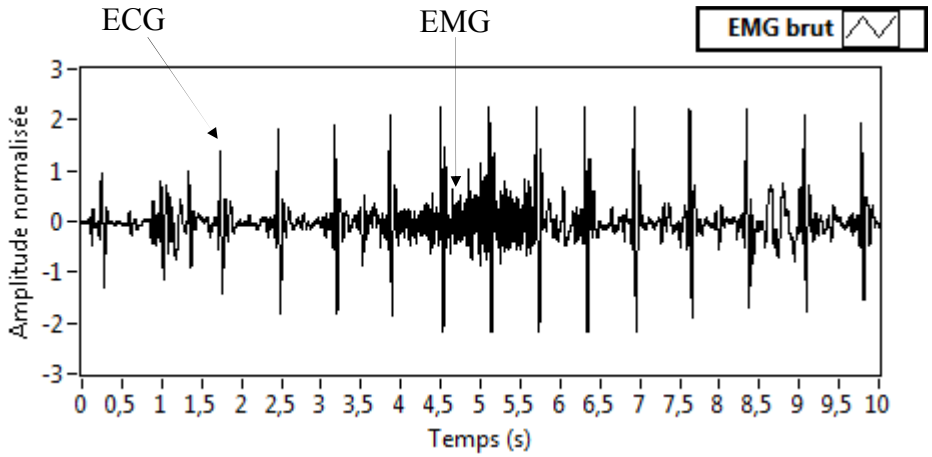


Figure 1.2 : Signal EMGdi contaminé par des ECG

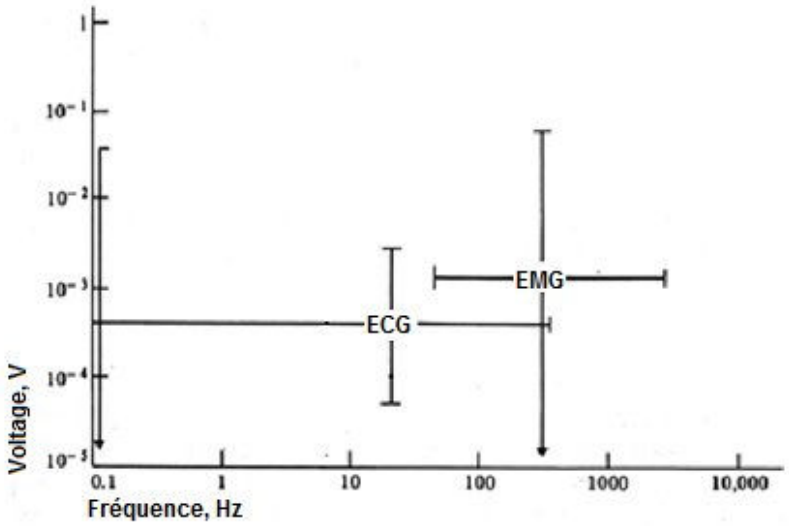


Figure 1.3 : Intervalles de fréquence et d'amplitude des signaux EMG et ECG [Webster, 1998]

Ce chapitre présente les méthodes principales d'élimination des EMG dans les ECG en exposant brièvement les avantages et les inconvénients de chaque méthode. Ceci permettra d'aborder le second chapitre de ce mémoire dans lequel sera proposée une étude comparative des différentes méthodes présentées dans ce chapitre.

Ce chapitre expose également les méthodes existantes de détection de l'activation/désactivation musculaire en se basant sur l'EMG. Ceci permettra en effet de connaitre les techniques pouvant être appliquées dans le cadre du projet de la réalisation du système d'acquisition des signaux respiratoires décrit précédemment. Ceci permettra également de connaitre les avantages offerts par la nouvelle technique de détection de l'activité musculaire du diaphragme qu'on proposera dans le troisième chapitre de ce mémoire.

1.2 Principales méthodes d'élimination des ECG dans les EMG

Depuis le développement des techniques d'acquisition des signaux EMG, plusieurs méthodes d'élimination des ECG dans les EMG ont été proposées. On pourrait regrouper ces techniques de filtrage en sept catégories : le filtrage par coupage [Schweitzer *et al.*, 1979], le filtrage par soustraction [Bartolo *et al.*, 1996; Bloch, 1983; Levine *et al.*, 1986], le filtrage adaptatif [Akkiraju et Reddy, 1992; Marque *et al.*, 2005; Widrow *et al.*, 1975; Zhou et Kuiken, 2006], le filtrage utilisant la transformée en ondelettes [Donoho et Johnstone, 1994; Zhou et Kuiken, 2006], le filtrage spectral [Redfern *et al.*, 1993], le filtrage basé sur la méthode analyse de composante indépendante (ICA) [Cao *et al.*, 2005; Hu *et al.*, 2007] ainsi que le filtrage regroupant plusieurs modules de filtrage en série [Zhou *et al.*, 2007], qu'on appellera par la suite «filtrage hybride» ou « technique hybride ».

Dans les paragraphes qui suivront, on expliquera plus en détails ces techniques de filtrage tout en montrant les avantages et les inconvénients de chaque catégorie.

1.2.1 Filtrage par coupage

Cette méthode consiste à sélectionner et à mettre à zéro chaque onde du signal ECG. Le fait que les amplitudes des signaux ECG détectés par les électrodes d'acquisition des signaux EMG sont, en général, supérieures aux amplitudes des signaux EMG détectés est exploité afin de distinguer les signaux ECG des signaux EMG.

Le principe de fonctionnement de cette technique est simple : un seuil choisi par l'opérateur est fixé et tout signal dépassant ce seuil est considéré comme un signal ECG et est remis à zéro. Dans certaines applications de cette méthode, les pics des ECG sont détectés et les intervalles se trouvant à une certaine durée avant et après chaque pic sont déterminés et les parties des signaux se trouvant dans ces intervalles sont également mises à zéro afin d'éliminer les composantes QRS entières du signal ECG. En effet, une composante QRS, définissant un intervalle du signal ECG contenant un battement de cœur entier, est composée de plusieurs segments cardiaques de caractéristiques différentes appelés segments Q, R et S et chacun de ces segments doit être éliminé lors du filtrage des EMG.

Cependant, puisque les ECG se trouvent parfois confondues avec les EMG et puisque les EMG peuvent parfois avoir une amplitude supérieure au seuil fixé, cette méthode peut engendrer des pertes dans des parties du signal EMG même en optimisant la valeur du seuil fixé. Néanmoins, cette méthode a comme avantage la facilité de sa mise en œuvre et la rapidité relative de son temps de traitement.

1.2.2 Filtrage par soustraction

Cette technique a été développée en 1983 dans [Bloch, 1983] afin de proposer une alternative à la technique de coupage présentée dans la section précédente. En effet la technique de coupage peut créer des pertes dans les signaux EMG lors de l'élimination des ECG car, une fois le signal ECG détecté par la technique de coupage, sa séquence est systématiquement remise à zéro même si elle est mélangée avec des signaux EMG.

La technique de filtrage par soustraction exploite le fait que les signaux ECG sont quasi périodiques et que, durant la période d'expiration, il y a une absence du signal EMG et le bruit est très faible. Ainsi, pendant cette période, on peut obtenir un signal qui ne contient quasiment que des signaux ECG. Par la suite, les signaux ECG contenus dans ce signal composé d'ECG sont extraits un à un et sont utilisés pour faire des opérations sur les signaux EMG non filtrés.

La première étape consiste en la formation d'un groupe contenant les modèles des signaux ECG pris pendant la période d'expiration. Les signaux de chaque modèle sont par la suite superposés en un seul signal. On obtient ainsi un groupe de signaux ECG représentant la moyenne des signaux ECG pendant une expiration. Dans chaque signal de ce groupe, deux parties sont distinguées : une partie destinée à des opérations de corrélation utilisée pour détecter la contamination ECG dans les EMG, et une partie destinée aux opérations de soustraction des ECG dans les EMG. Ainsi, lorsqu'une contamination ECG est détectée en utilisant les signaux de corrélation, une opération de soustraction entre le signal EMG contaminé et les signaux de la partie destinée aux opérations de soustraction est réalisée afin d'obtenir un signal EMG filtré.

La méthode de soustraction a connu différents développements pour améliorer ses performances [Bartolo et al., 1996; Levine et al., 1986] tout en gardant le même principe de fonctionnement qu'on a présenté précédemment. Cette méthode, malgré les bons résultats qu'elle donne, reste limitée car elle nécessite l'acquisition séquentielle des EMG contaminés et des ECG pures afin d'effectuer l'opération de soustraction.

1.2.3 Transformée en ondelettes

La transformée en ondelettes continue est une transformée utilisée pour décomposer une fonction continue dans le temps en ondelettes. À la différence de la transformée de Fourier qui ne permet qu'une décomposition en fréquence d'une fonction donnée, la transformée en ondelettes permet de décomposer une fonction en fréquence et en temps. La transformée en ondelettes est exprimée par la fonction suivante :

$$\mathrm{CWT}(a,b) = \frac{1}{\sqrt{a}} \int x(t) \psi\left(\frac{t-b}{a}\right) dt \qquad (1.1)$$

où a ∈ R+ est un facteur de compression, b ∈ R est un facteur de translation, x est la fonction d'entrée et ψ est l'ondelette mère.

Grâce aux avantages que la transformée en ondelettes offre par rapport à la transformée de Fourier, cette méthode est considérée comme un outil efficace pour extraire des signaux à partir d'enregistrements contaminés par du bruit [Donoho et Johnstone, 1994]. Zhou et Kuiken [Zhou et Kuiken, 2006] utilisent la transformée en ondelettes comme méthode pour éliminer le bruit cardiaque dans les signaux EMG. Les étapes de traitement utilisées y ont été décrites comme suit : le signal EMG brut est tout d'abord décomposé, en utilisant la transformée en

ondelettes, en composantes basse-échelle et hautes fréquences d'un côté, et en composantes haute-échelle et basses fréquences d'un autre. Ces dernières composantes, considérées comme contenant la contamination par les ECG, sont par la suite traitées par un processus de « seuillage ». Le signal final, représentant le signal contenant les EMG traités, est reconstruit à partir des nouvelles composantes en utilisant la transformée en ondelettes inverse.

L'application de cette méthode a permis d'éliminer une grande partie des ECG sans pertes significatives dans les EMG [Zhou et Kuiken, 2006]. Cependant, le temps nécessaire pour le traitement des données par cette méthode ne permet pas l'utilisation de cette méthode en temps-réel [Zhou *et al.*, 2007].

1.2.4 Méthode ICA

La méthode ICA, appliquée au problème d'élimination des ECG contaminant les EMG, consiste en la séparation des signaux EMG et des signaux ECG à partir d'un signal EMG contaminé par des ECG. Le but recherché est de trouver une matrice de transformation A, tel que : x = As, avec x est la transposée de la matrice [x1,x2,…,xm] représentant les données en entrée (EMG contaminé), s est la transposée de la matrice [s1,s2,…,sm] représentant les données de sortie (EMG filtré). Plusieurs algorithmes pour déterminer la matrice A ont été développés tel que les algorithmes FastICA, JADE et infomax. Dans [Hu *et al.*, 2007], l'utilisation de l'algorithme JADE a permis d'éliminer une grande partie de la contamination ECG sans pertes considérables dans les signaux EMG.

L'application de la méthode ICA, malgré son efficacité, reste limitée car elle ne peut être utilisée dans des applications opérant en temps réel à cause de son délai de traitement dû à la

complexité des algorithmes pouvant être utilisés dans le cadre de cette méthode. Plus encore, afin d'améliorer les performances d'un prétraitement fréquentiel nécessaire avant l'application de l'algorithme ICA, il est conseillé d'utiliser un canal d'acquisition des ECG afin d'évaluer la limite fréquentielle contenant plus de 95% de la densité de puissance spectrale des signaux ECG acquis via ce canal [Hu *et al.*, 2007].

1.2.5 Le filtrage spectral

Cette technique de filtrage bénéficie du fait que la plus grande partie de la puissance spectrale des signaux ECG est localisée dans les bandes basses fréquences. Ainsi, un filtrage passe-haut adéquat pourrait éliminer une grande partie des signaux ECG. Le filtre passe-haut le plus utilisé pour cette fonction est le filtre de type Butterworth.

Ce type de filtrage est le plus facile à réaliser et le moins coûteux et peut être utilisé pour un filtrage en temps réel. Plusieurs études ont été faites pour choisir les paramètres du filtre (ordre et fréquence de coupure) permettant d'avoir des performances optimales. L'une des études les plus importantes sur ce sujet a été réalisée par Redfern *et al.* [Redfern *et al.*, 1993] qui est arrivée à la conclusion que la fréquence de coupure optimale à utiliser pour un filtre passe-haut de Butterworth pour le filtrage des EMG est de 30 Hz. Concernant l'ordre optimal de ce filtre, la plupart des études s'accordent sur le fait que l'ordre optimal est égal à 4.

L'inconvénient de cette méthode est sa performance moyenne. En effet, ce filtrage permet d'atténuer la contamination ECG sans pour autant atteindre l'efficacité nécessaire pour certaines applications.

1.2.6 Le filtrage adaptatif

Le filtrage adaptatif est une méthode efficace pour éliminer un signal d'interférence dans un autre signal lorsque ces deux signaux ont des spectres de fréquences non distincts [Widrow *et al.*, 1975]. Cette technique nécessite un signal de référence corrélé avec la source du bruit (dans notre cas avec les signaux ECG), ce qui ajoute une contrainte supplémentaire car la mise en place d'un canal supplémentaire d'acquisition des ECG devient nécessaire pour l'acquisition du signal à corréler. Cependant, cette méthode de filtrage donne de bons résultats pour l'élimination des ECG tout en évitant la perte de parties importantes des EMG après le filtrage [Chen *et al.*, 1994; Widrow *et al.*, 1975; Zhou et Kuiken, 2006].

Le principe d'un filtre adaptatif est présenté sur la figure 1.4. Ce filtre reçoit en entrée les signaux EMG contaminés ainsi que les signaux ECG bruts acquis simultanément en utilisant une seconde source d'acquisition. Dans la figure 1.4, z^{-1} représente un délai d'un échantillon, j la constante de temps et les coefficients w des coefficients d'ajustements. L'équation représentant la sortie en fonction y en fonction de l'entrée n est la suivante [Zhou et Kuiken, 2006]:

$$yj = \sum_{i=1}^{N} w_{i,j} n_{1,j-1+1} \tag{1.2}$$

Dans [Chen et al., 1994], il a été démontré, en utilisant des EMGdi acquis via des électrodes insérées dans l'œsophage, qu'en minimisant l'erreur quadratique moyenne, le filtre adaptatif de la figure 1.4 pouvait produire un signal yi représentant une réplique exacte des signaux ECG contaminant l'EMG de l'entrée et pouvait, par conséquent, produire une sortie contenant des EMG filtrés des contaminations ECG. Une application sur des signaux EMG réels obtenus sur des

muscles thoraciques en minimisant l'erreur quadratique moyenne précédente a également été validée par Zhou-Kuiken [Zhou et Kuiken, 2006]. Cependant, cette technique de filtrage, même si elle donne de bons résultats, nécessite un temps important de traitement qui ne permet pas son utilisation pour des applications opérant en temps réel [Zhou *et al.*, 2007].

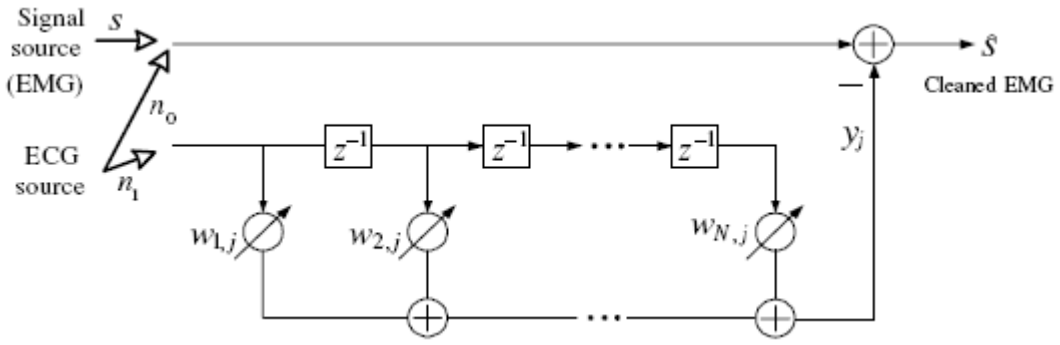


Figure 1.4 : Structure d'un filtre adaptatif pour l'élimination des ECG [Zhou et Kuiken, 2006]

1.2.7 Le filtrage hybride

1.2.7.1 Description générale

Cette méthode est inspirée du filtrage des signaux EMG du muscle thoracique pour une utilisation dans une prothèse myoélectrique [Zhou *et al.*, 2007]. Cette procédure est une combinaison de deux dispositifs exécutés en série :

- Une approche basée sur l'algorithme « coupure du pic » (ou « spike-clipping » en anglais) qu'on notera SC par la suite [Zhou *et al.*, 2007].

- Une technique de filtrage en utilisant un filtre passe-haut de Butterworth.

L'utilisation du filtre de Butterworth passe-haut seul n'élimine pas tous les signaux ECG se trouvant dans le signal EMG car, comme expliqué précédemment, les spectres de l'EMG et de l'ECG ont une zone d'intersection.

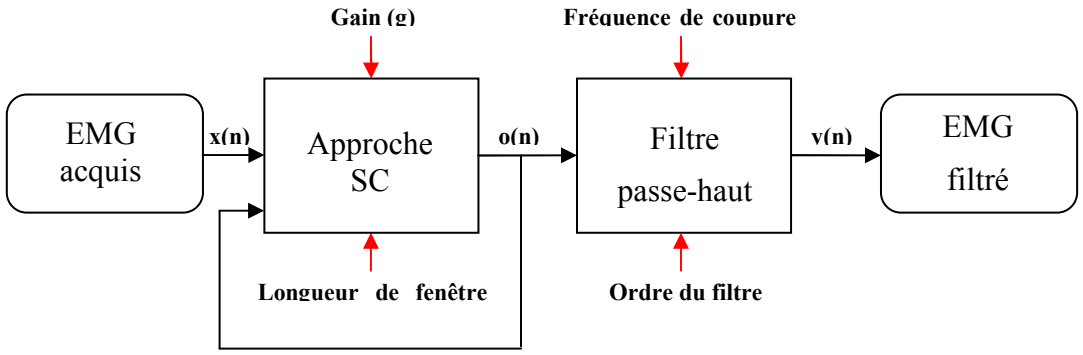


Figure 1.5 : Schéma bloc de la technique hybride

Ainsi, afin d'éliminer les signaux ECG dont le spectre est confondu avec celui de l'EMG, l'algorithme SC est appliqué en série avec un filtre Butterworth passe-haut (figure 1.5). Dans ce qui suit, plus de détails seront donnés sur les deux techniques composant la méthode hybride. Les algorithmes faisant partie de cette technique seront présentés dans le prochain chapitre.

1.2.7.2 Approche « coupure du pic »

Les algorithmes basés sur le filtrage par coupage, comme nous l'avons montré dans la section 1.2.1, utilisent des seuils fixes afin de détecter les signaux EMG contaminant les signaux EMGdi. Cependant, un seuil fixe de faible valeur peut être assez bas pour éliminer l'ECG dans le cas de repos mais va couper l'EMG en cas de contraction musculaire et un seuil fixe haut risque de ne pas détecter une grande partie de la contamination ECG. Ainsi, l'approche SC propose un

algorithme à seuil variable dont la valeur dépend d'une moyenne calculée d'une manière itérative à chaque point du signal EMG. Deux paramètres, qui seront expliqués dans le prochain chapitre, permettent de régler les performances de cet algorithme. Des valeurs optimales de ces paramètres doivent être choisies pour éliminer le maximum de signaux ECG tout en minimisant les pertes dans les signaux EMGdi.

1.2.7.3 Le filtre passe-haut

Comparé au spectre de l'EMG, la plupart de la puissance de l'ECG est localisée relativement dans les bandes basses fréquences. Une utilisation d'un filtre de Butterworth pourrait éliminer ces ECG dont le spectre est en dehors de celui des EMG. Ceci peut se faire en choisissant des valeurs optimisées de la fréquence de coupure et de l'ordre de ce filtre.

1.3 Méthodes de détection de l'activité musculaire

1.3.1 Seuillage classique

Ces méthodes de détection de l'activité musculaire à partir de l'EMG consistent à fixer un seuil optimal et à considérer tout signal dépassant ce seuil comme une activité musculaire. Ainsi, le signal de sortie prend un état actif lorsque le signal EMG acquis dépasse la valeur fixée du seuil. La sortie reste active pour une durée prédéterminée, notée T, après un dépassement de la valeur du seuil. Si pendant cette durée, le signal d'entrée dépasse encore une fois la valeur du seuil, l'attente T est remise à zéro et le signal de sortie reste actif encore une fois pour la durée T. Dans certaines applications de cette méthode, afin d'éviter l'utilisation du temps d'attente T,

le signal EMG est rectifié, puis filtré par un filtre passe-bas avant d'être comparée directement au seuil fixé sans avoir à attendre pendant une durée t avant de désactiver la sortie ou la laisser activée [Zecca *et al.*, 2002].

Plusieurs études ont démontré expérimentalement le fonctionnement de ces méthodes [Chabot *et al.*, 2006; Mrvaljevic *et al.*, 2007; Zecca *et al.*, 2002]. Les techniques de seuillage classiques sont les plus simples à mettre en œuvre et les moins couteuses. Cependant, la qualité de la détection par ces techniques reste limitée à cause de leur caractère statique. Ainsi, un seuil de valeur basse est sensible au bruit, ce qui peut donner en sortie de fausses détections, alors qu'un seuil élevé est moins précis car il peut ne pas détecter certaines faibles activations musculaires.

L'utilisation de ces méthodes implique la mise en place d'un prétraitement du signal EMG acquis afin de ne pas confondre les différentes sources de bruit affectant le signal EMG acquis et l'activité musculaire.

1.3.2 Algorithme itératif basé sur l'EMGdi pour la détection de l'inspiration

Cet algorithme, proposé par Dow *et al.* [Dow *et al.*, 2006], a pour objectif de détecter l'activation du diaphragme à partir de l'EMGdi afin d'être utilisé dans des applications de contrôle de la ventilation mécanique. Cet algorithme peut être synthétisé par les trois points suivants:

1- Le signal EMGdi acquis est tout d'abord conditionné en échantillonnant ce signal et en éliminant l'offset, puis en diminuant l'effet du bruit en fixant un seuil r et en limitant toute valeur du signal dépassant ce seuil à la valeur r selon la formule :

$$zi = \begin{cases} yi \leq r : yi \\ yi > r : r \end{cases} \quad (1.3)$$

où yi est le signal EMGdi échantillonné et zi le signal résultant après la comparaison avec le seuil r.

Un seuil, noté t, qui sera utilisé plus loin dans l'algorithme, est aussi évalué dans cette étape par l'équation :

$$t = \frac{\sum_{i=0}^{n} yi}{n} \quad (1.4)$$

2- Le calcul de la moyenne mobile finie (MMF) : à chaque itération, une moyenne wi, où i est l'indice de l'échantillon EMGdi actuel, est calculée à partir des m points de l'EMGdi échantillonné contenus dans une fenêtre limitée par les points d'indice i-m+1 et d'indice i. L'équation donnant cette moyenne est la suivante :

$$wi = \frac{\sum_{k=i-m+1}^{i} zk}{m} \quad (1.5)$$

3- À chaque itération de l'algorithme, si wi ≤ t : l'algorithme considère qu'il n'y a pas d'inspiration et un compteur, noté C, est remis à zéro. Pour considérer qu'il y a une inspiration, la moyenne wi doit rester au dessus de t pour une certaine période de temps prédéterminée et calculée par le compteur C qui s'incrémente à chaque fois que wi est au dessus de t. Cependant, si wi tombe en-dessous du seuil t avant d'atteindre le temps nécessaire pour considérer la portion actuelle du signal comme une inspiration, le compteur C est remis à zéro.

Cet algorithme a été testé [Dow *et al.*, 2006] en l'appliquant sur des signaux EMGdi réels pris sur un rat de laboratoire. Les résultats ont démontré que, dans le cas où le rat de laboratoire est en état de repos, plus de 99% des détections ont été détectées et moins de 1 % des détections étaient fausses. Dans le cas où le rat de laboratoire est agité, 81% des inspirations ont été détectées et 6 % des détections étaient fausses.

1.3.3 Méthodes de seuillage avancées

1.3.3.1 Description générale

Les méthodes de seuillage avancées, présentées dans de nombreuses publications [Abbink *et al.*, 1998; Bonato *et al.*, 1998; Hodges et Bui, 1996; Lidierth, 1986], reposent sur le schéma de détection des événements de la figure 1.6 [Straude *et al.*, 2001; Straude et Wolf, 1999].

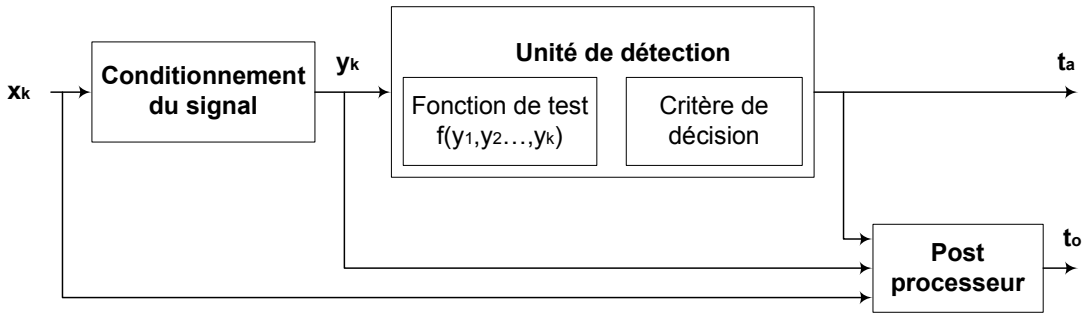


Figure 1.6 : Schéma de détection des événements dans les signaux EMG

En premier lieu le signal EMG numérisé, noté x_k, est traité par un filtre passe-bas afin d'éliminer le bruit haute-fréquence. Ensuite, le signal EMG conditionné, noté y_k, est envoyé à une unité de détection afin d'évaluer un temps d'alarme, noté t_a, indiquant l'instant d'une possible activation de l'EMG. Le temps d'alarme est évalué par le biais d'une fonction de test spécifique à chaque méthode. Le critère de décision permettant d'évaluer t_a est défini par la relation :

$t_a = \min \{k \geq 1 : f(y_1, y_2, \ldots, y_k) \geq h\}$ (1.6)

où h est un seuil prédéfini.

La fonction de test f est évaluée par rapport au point présent, y_k, ainsi que des données EMG des instants précédents. L'évaluation de f se fait continuellement tant que la valeur actuelle de f est inférieure au seuil h. Lorsque la valeur de f dépasse le seuil h, le calcul de f est arrêté et la valeur de t_a est fixée à l'instant où le dépassement du seuil s'est produit. À cet instant, le post processeur est déclenché afin d'évaluer le temps estimé de déclenchement, noté t_o. Dans la plupart des méthodes de détection proposées, le post processeur prend en entrée uniquement le temps d'alarme t_a et le signal conditionné y_k.

La fonction de test f, le critère de décision, ainsi que le post processeur diffèrent d'une méthode à l'autre. Dans les paragraphes qui suivent, on présentera les méthodes de détection les plus connues basées sur le schéma de détection décrit précédemment.

1.3.3.2 Méthodes de Hodges et Lidierth

Les méthodes proposées par Hodges et Lidierth [Hodges et Bui, 1996; Lidierth, 1986] ont des fonctions de test et des critères de décision identiques et diffèrent uniquement sur le post processeur utilisé. Ces méthodes évaluent la fonction de test f en se basant sur un algorithme de moyenne mobile finie (MMF). Ainsi, la fonction de test f est calculée en moyennant N échantillons contenus dans une fenêtre mobile dont les deux extrémités sont l'échantillon actuel, y_k, et l'échantillon y_{k-N+1}. La fonction f est définie par la relation suivante à tout instant k:

$$f(y1, y2, \ldots, yk) = \frac{1}{\sigma o}\left[\left(\frac{1}{N}\sum_{i=k-N+1}^{k} yi\right) - \mu o\right] \quad (1.7)$$

où σo et μo sont l'écart type et la moyenne des M premiers échantillons du signal y_k.

Le critère de détection utilisé dépend de la largeur de la fenêtre choisie et il est défini par la relation :

$$t_a = \min \{k \geq N : f(y_1, y_2, \ldots, y_k) \geq h\} \quad (1.8)$$

Hodges et Bui [Hodges et Bui, 1996] proposent un post processeur simple. En effet, **l**e temps de déclenchement, t_o, est estimé en fonction du temps d'alarme et de la largeur de fenêtre de la MMF par la relation suivante :

$$t_o = t_a - N + 1 \quad (1.9)$$

Cependant, Lidierth [Lidierth, 1986] propose un post processeur différent. Ce post processeur accepte une activation de l'EMG si la fonction de test f est supérieure au seuil h pour N1 échantillons et que, durant ces N1 échantillons, la fonction f ne peut descendre en-dessous du seuil h pour plus de N2 échantillons.

La méthode introduite par Hodges et Bui [Hodges et Bui, 1996] a été implémentée en fixant les paramètres h, M et N aux valeurs suivantes : h = 2.5, M = 200, N = 50. Dans [Lidierth, 1986], les valeurs des paramètres utilisés ont été fixées à : N1 = 90, N2 = 15 et h = 3. Cependant, ces valeurs peuvent varier selon l'amplitude du signal, la fréquence d'échantillonnage utilisée, l'écart type, ainsi que de la précision désirée et de la netteté du signal EMG.

1.3.3.3 Méthode de Bonato

Dans la méthode décrite Bonato *et al.* [Bonato *et al.*, 1998], la fonction de test est calculée, pour les valeurs impaires de k uniquement, par la relation :

$$gk = f(y1, y2, \ldots, yk) = \frac{1}{\sigma o^2}\left(y_{k-1}^2 + y_k^2\right) \quad (1.10)$$

où σo est l'écart type des M premiers échantillons du signal y_k.

Le critère de détection dépend de y_k uniquement et il est évalué par l'équation :

$$t_a = \min \{k = 1 , 3 , 5 , \ldots : g_k \geq h\} \quad (1.11)$$

Le post processeur accepte un déclenchement si et seulement si au moins n des m échantillons successifs dépassent le seuil h afin de considérer l'état de déclenchement comme actif et que l'état de déclenchement actif dure N échantillons. Dans ce cas, l'instant correspondant au premier échantillon de l'état de déclenchement actif est considéré comme étant la valeur du temps t_o.

Le choix des paramètres h, n, m, M et N dépend de l'écart type, de la fréquence d'échantillonnage, l'amplitude du signal, ainsi de la précision désirée et de la netteté du signal EMG utilisé.

1.4 Conclusion

Dans ce chapitre, nous avons exposé les méthodes les plus connues et les plus utilisées concernant l'élimination des ECG. Dans le chapitre qui suit, une étude comparative de ces méthodes sera faite en se basant sur les avantages et les inconvénients de chaque méthode et en choisissant les méthodes pouvant être intégrées au système présenté dans la section 1.1.

Ce chapitre a également exposé les méthodes principales de détection de l'activité musculaire en se basant sur les signaux EMG. Ces méthodes seront comparées dans le chapitre suivant afin de choisir les méthodes pouvant opérer dans le système présenté dans la section 1.1.

CHAPITRE 2 ÉTUDE COMPARATIVE DES TECHNIQUES D'ÉLIMINATION DES ECG DANS LES EMG ET DE LA DÉTECTION DE L'ACTIVITÉ MUSCULAIRE

2.1 Introduction

Nous présentons dans ce chapitre une comparaison des méthodes d'élimination de la contamination ECG dans les signaux EMG, présentées dans le chapitre précédent, afin de trouver les techniques pouvant être utilisées pour opérer en temps réel et d'une manière autonome. Les méthodes choisies seront comparées en simulation en utilisant des signaux EMGdi réels acquis via le système d'acquisition mis en œuvre par [Désilets *et al.*, 2006]. Ce chapitre présente également une comparaison des méthodes de détection de l'activité musculaire afin de choisir la méthode la plus adaptée à notre application. Étant donné qu'une meilleure détection de l'activité musculaire se fait sur des signaux EMG avec le minimum de contamination ECG, la méthode choisie ne sera testée qu'après avoir proposé notre méthode d'élimination des ECG dans le chapitre qui suit afin d'avoir des résultats plus significatifs. La méthode de détection de l'activité musculaire choisie sera également adaptée afin d'être compatible avec notre application.

2.2 Étude comparative des méthodes d'élimination des ECG dans les EMG

2.2.1 Comparaison et choix des méthodes compatibles

Dans ce paragraphe, les méthodes d'élimination de la contamination ECG dans les signaux EMG seront comparées pour choisir les méthodes pouvant opérer dans le système présenté dans [Désilets et al., 2006]. Ainsi, ces méthodes doivent satisfaire aux critères suivants :

1. Être capables d'opérer en temps-réel.
2. Être capables d'opérer d'une manière autonome et automatique.
3. Utiliser uniquement le signal EMGdi brut comme entrée sans avoir besoin d'autres entrées nécessitant l'ajout de moyens supplémentaires pour l'acquisition d'autres signaux.

Comme cité dans la section 1.2 (premier chapitre), les méthodes d'élimination de la contamination ECG dans les signaux EMG sont divisées en sept catégories principales. Le tableau 2.1 résume, d'une manière générale, les caractéristiques de ces catégories.

Aussi, la liste des caractéristiques des méthodes est:

1 : Élimination d'une grande partie de la contamination par ECG.

2 : Technique simple à mettre en œuvre.

3 : Possibilité d'utilisation dans une application temps-réel.

4 : Peu de pertes dans les signaux EMG après le filtrage.

5 : Technique pouvant opérer d'une manière autonome.

6 : Ne nécessite pas un canal supplémentaire pour l'acquisition des ECG.

Tableau 2.1: Caractéristiques des diverses méthodes d'élimination des ECG dans les EMG

Méthode	**Caractéristiques**					
	1	**2**	**3**	**4**	**5**	**6**
Filtrage par coupage		X	X		X	X
Filtrage par soustraction	X			X		
Transformée en ondelettes	X			X	X	X
Méthode basée sur l'ICA	X			X		X
Filtrage passe-haut		X	X		X	X
Filtrage adaptatif	X			X	X	
Filtrage hybride	X		X	X	X	X

Parmi les méthodes présentées précédemment, le filtrage par coupage, le filtrage passe-haut et la méthode hybride sont les seules techniques satisfaisant ces conditions. Le choix définitif de la technique à adapter reposera donc essentiellement sur les performances de chaque méthode.

Dans le passé, aucune comparaison expérimentale n'a été faite entre ces trois méthodes en utilisant des signaux EMGdi réels bien qu'une comparaison entre le filtrage passe-haut et la technique hybride ait été faite dans [Zhou *et al.*, 2007] sur des signaux EMG thoraciques et a démontré la supériorité de la technique hybride par rapport au filtrage passe-haut. Cependant, les résultats de cette comparaison ne peuvent pas être considérés comme applicables sur les signaux EMGdi car les signaux EMG thoraciques utilisés étaient contaminés par des composantes ECG hautes fréquences ce qui n'est pas nécessairement le cas des signaux EMGdi qui nous intéressent. Ainsi, on ne pourrait valider sans simulation sur des signaux réels EMGdi le fonctionnement de la technique hybride dans le cadre de notre application.

2.2.2 Programme implémentant les méthodes choisies

Afin de comparer les trois méthodes retenues dans les paragraphes précédents et valider la technique hybride en utilisant des EMGdi réels, on a implémenté ces trois méthodes en langage LabView. Les programmes réalisés permettent de contrôler les paramètres d'entrée de ces méthodes, d'afficher et de sauvegarder les résultats obtenus par le biais d'une interface graphique.

Dans les paragraphes qui suivent, on présentera en détail les méthodes implémentées.

2.2.2.1 Technique Hybride

Cette technique de filtrage est composée, comme on l'a exprimé dans le chapitre précédent, de deux techniques de filtrage en série : la première technique de filtrage est l'algorithme SC et la seconde est un filtre de Butterworth passe-haut.

L'algorithme représentant la première technique de filtrage (SC) est un algorithme basé sur le calcul d'une moyenne rectifiée à chaque itération. Cet algorithme est décrit par les étapes suivantes :

- On fixe la valeur d'un paramètre l = NT avec T la période d'échantillonnage et N un certain nombre de points échantillonnés.
- On appellera g le gain de l'algorithme. La valeur de ce gain est fixée par l'utilisateur.
- On considère que x(t) représente le signal EMGdi contaminé par les ECG et o(t) le signal EMGdi après le filtrage par SC.
- Pour les N premiers points échantillonnés, le signal reste le même, c'est-à-dire que x(t) = o(t) pour t = 1,...,N. On calcule dans cette fenêtre de points une somme, appelée ARV (Averaged Rectified Value ou valeur moyenne corrigée). Cette somme est exprimée par la relation suivante [Zhou *et al.*, 2007] :

  $$\text{ARV}(1) = \frac{1}{N}\sum_{k=1}^{N}|o(k)| = \frac{1}{N}\sum_{k=1}^{N}|x(k)| \quad (2.1)$$

- Pour le point N + 1 : si $|x(N+1)| > g \times \text{ARV}(1)$, alors on pose :

  $$o(N+1) = \text{sgn}(x(N+1)) \times g \times \text{ARV}(1) \quad (2.2)$$

  où sgn est la fonction signe.

  Si $|x(N+1)| \leq g \times \text{ARV}(1)$ alors on pose :

$o(N+1) = x(N+1)$ (2.3)

- Par la suite, on recalcule la somme ARV :

$ARV(2) = \frac{1}{N}\sum_{k=2}^{N+1}|o(k)|$ (2.4)

- Ce procédé est répété à chaque fois. Ainsi, pour l'instant d'échantillonnage N+m, où m = 2,3,…,L (où N + L est la longueur du signal) :

si $|x(N+m)| > g \times ARV(m)$, alors on pose :

$o(N+m) = sgn(x(N+m)) \times g \times ARV(m)$ (2.5)

Sinon :

$o(N+m) = x(N+m)$ (2.6)

- La somme ARV sera calculée à chaque étape par l'expression :

$ARV(m) = \frac{1}{N}\sum_{k=m}^{N+m-1}|o(k)|$ (2.7)

Cette technique permet la détection dynamique des pics des ECG ainsi que des artéfacts de mouvement des électrodes et permet de les atténuer considérablement comme on va le constater dans les résultats de simulation.

Cet algorithme est mis par la suite en série avec un filtre de Butterworth passe-haut en ajustant bien l'ordre de ce filtre ainsi que sa fréquence de coupure afin d'éliminer les signaux ECG restant.

Quatre modules composants cette méthode ont été implémentés en langage LabView (figure 2.1). Le premier module est chargé d'extraire l'enregistrement EMGdi à partir d'un fichier texte

et mettre les données en forme avant de les envoyer au module suivant. Ce dernier module traite ces données par l'algorithme SC présenté précédemment puis les envoie au module de filtrage par le filtre passe-haut de Butterworth avant d'afficher la courbe EMGdi finale. Le module « sauvegarde » reçoit le signal EMGdi filtré et le stocke dans un fichier texte.

Les paramètres des modules de ce programme peuvent être définis dans une interface graphique qui permet également d'afficher la courbe EMGdi dans les diverses étapes de traitement (figure 2.2)[2].

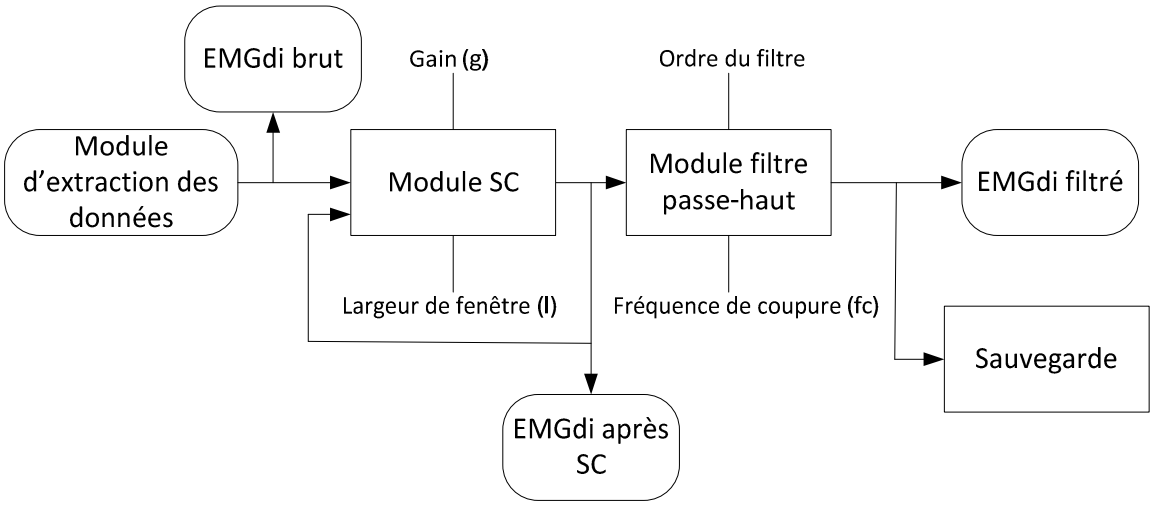


Figure 2.1 : Schéma bloc du programme implémentant la technique hybride

[2] Un logiciel regroupant toutes les interfaces présentées dans ce chapitre ainsi que des interfaces d'autres méthodes sera présenté dans le chapitre 3.

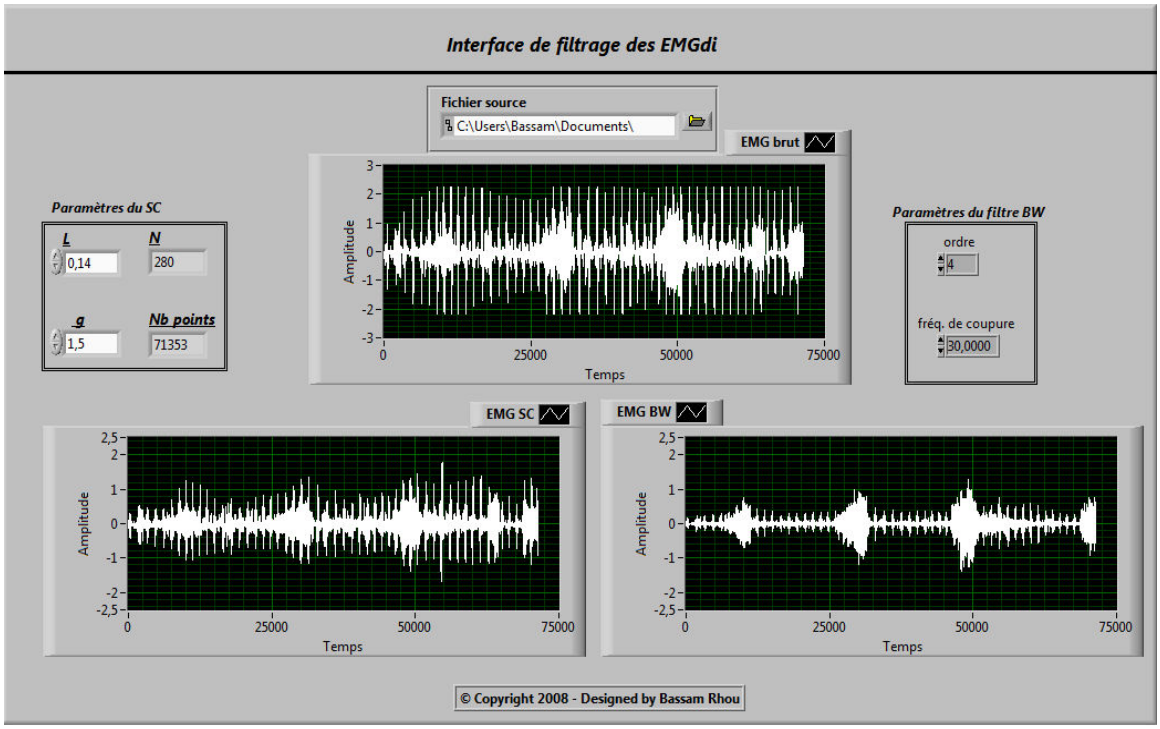


Figure 2.2 : Interface logicielle du programme de la technique hybride

2.2.2.2 Méthode de coupage

L'algorithme de coupage qu'on a implémenté opère de la manière suivante : à chaque itération, le point reçu à l'entrée du module de l'algorithme de coupage est comparé à la valeur du seuil. Si sa valeur est inférieure à la valeur du seuil, ce point est envoyé directement à la sortie du module pour être affiché, sinon, ce point est remis à zéro ainsi qu'un certain nombre de points reçus avant ce point. Un certain nombre de points EMG qui seront reçus après un point dépassant le seuil seront également remis à zéro. De cette façon, un segment du signal EMG, détecté après le dépassement du seuil fixé, est considéré comme une contamination ECG et est

remis à zéro. Le nombre de points à remettre à zéro est fixé de telle sorte que toutes les composantes QRS des signaux ECG soient remises à zéro.

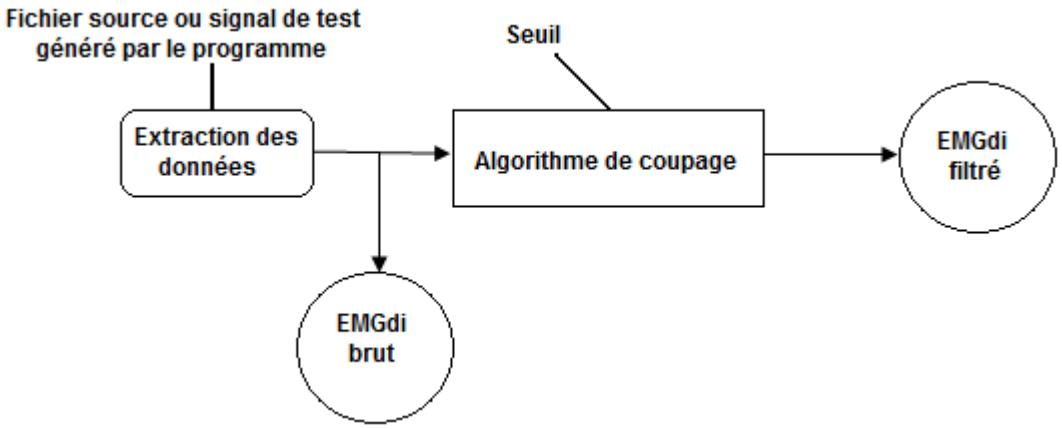


Figure 2.3 : Schéma bloc du programme implémentant l'algorithme de coupage

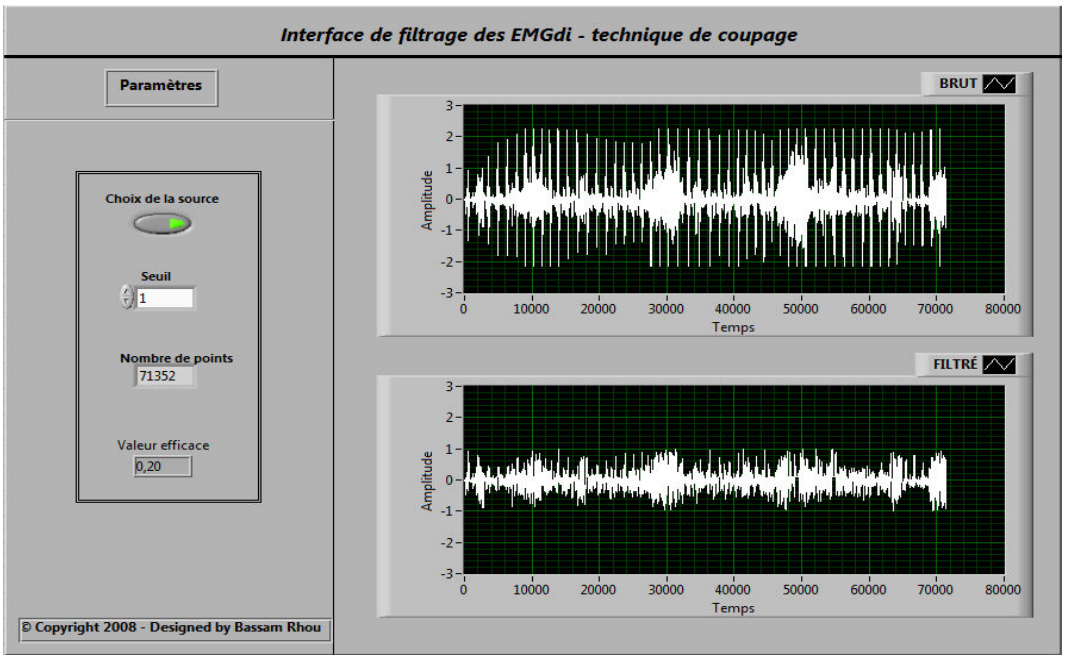


Figure 2.4 : Interface graphique du programme de la technique de coupage

Le programme réalisé est divisé en trois modules (figure 2.3): un module d'extraction des données, qui reçoit en entrée le chemin du fichier source ou un signal de test généré par le programme, un module implémentant l'algorithme de coupage, qui reçoit en entrée les points du signal envoyés par le module extraction de données ainsi que le seuil définit par l'utilisateur, et un module de sauvegarde des données qui stocke les données du signal résultant dans un fichier texte.

Les points en sortie du module d'extraction des données ainsi que les points en sortie de l'algorithme de coupage sont affichés dans les deux graphes de l'interface utilisateur (figure 2.4).

2.2.2.3 Filtre passe-haut

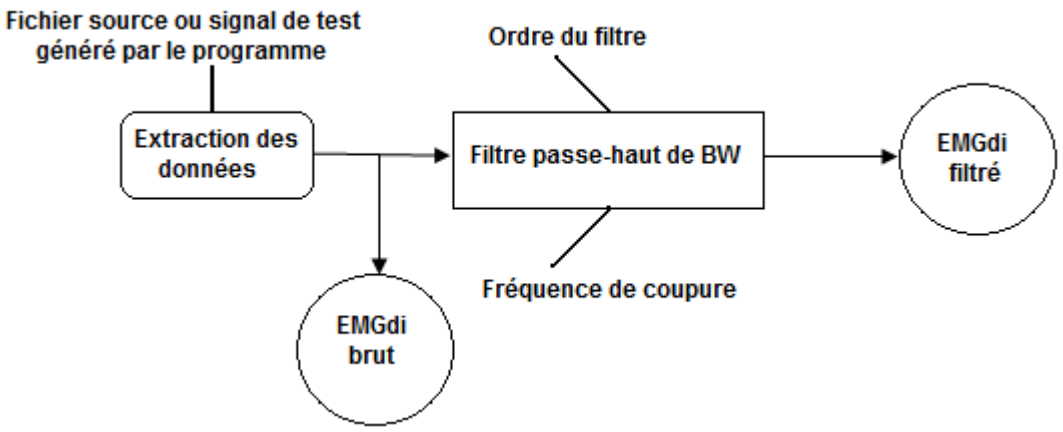


Figure 2.5 : Schéma bloc du programme implémentant le filtre passe-haut de BW

Ce filtre est le même filtre utilisé en série avec le module SC dans la technique hybride décrite dans la section 2.1.2.3. L'implémentation de ce filtre est simple puisque le logiciel LabView 8.5 offre déjà un bloc fonctionnel « filtre de Butterworth ». Les signaux EMGdi sont extraits par le module « extraction de données », similaire à celui utilisé dans la technique hybride, puis

envoyés au filtre passe-haut de Butterworth qui filtre le signal puis affiche le signal résultant et l'envoie au module qui sauvegarde le signal résultant dans un fichier texte (figure 2.5).

L'ordre de ce filtre ainsi que la fréquence de coupure peuvent être ajustés par le biais d'une interface graphique qu'on a réalisé (figure 2.6) afin de permettre d'effectuer facilement plusieurs tests pour trouver l'ordre et la fréquence de coupure permettant d'avoir les meilleurs résultats.

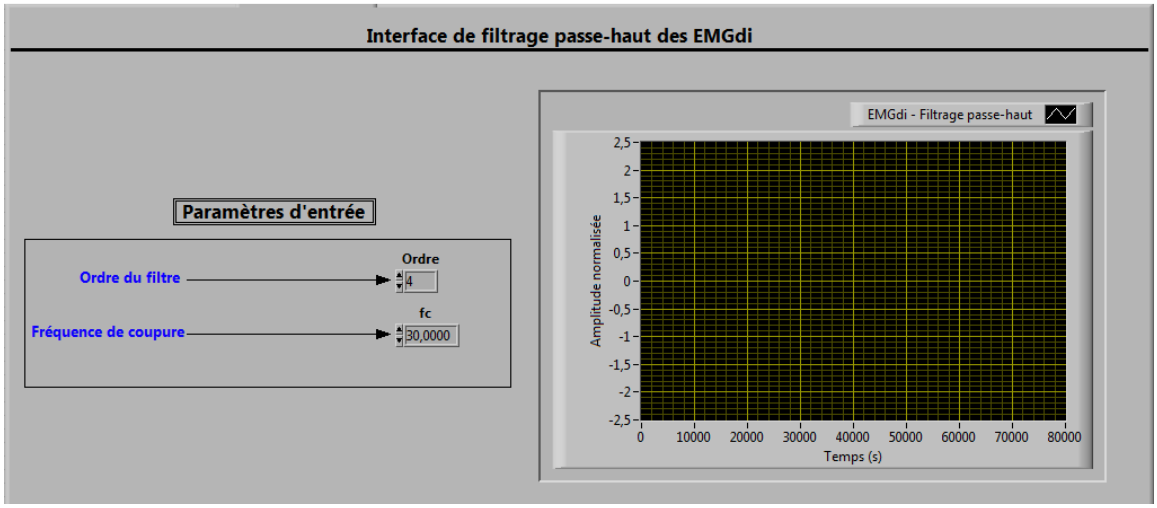


Figure 2.6 : Interface graphique du programme de filtrage passe-haut

2.2.3 Résultats obtenus

Les résultats de simulation obtenus en utilisant les programmes précédents sont montrés dans les figures 2.7, 2.8, 2.9 et 2.10. Les signaux utilisés sont des signaux EMGdi réel acquis sur une personne en bonne santé respiratoire via le système d'acquisition présenté dans le premier chapitre. Ces signaux ont été échantillonnés avec une fréquence de 2 kHz pendant une durée de

36 secondes. La respiration a été faite en retenant la respiration pendant plusieurs secondes afin de distinguer clairement les blocs EMGdi.

On peut remarquer que dans la courbe EMGdi traitée par la technique de coupage (figure 2.8), désignée également sous le nom « technique de seuillage », la perte des EMGdi est relativement importante et l'élimination des ECG n'est pas aussi efficace que dans le traitement par le filtrage passe-haut (figure 2.9) et la technique hybride (figure 2.10). D'autre part, on peut remarquer que la technique hybride élimine le plus de la contamination ECG tout en permettant de maintenir une conservation des EMGdi similaire à celle offerte par le filtrage passe-haut.

Le tableau 2.2 montre une comparaison entre les diverses performances des trois méthodes précédentes. Étant donné que la méthode d'élimination des ECG qu'on recherche est destinée à des applications temps-réel pouvant éliminer une grande quantité de contamination et pouvant être utilisée pour des applications de contrôle de la ventilation mécanique, on a choisi de comparer trois indicateurs de performance. Le premier indicateur choisi est la vitesse de traitement qui est important dans les applications temps-réel. Le second indicateur choisi est l'indice ARV (équation 2.9) qui est un bon indicateur des changements d'amplitude du signal. Le troisième indicateur choisi est le rapport signal/bruit (SNR) qui est un paramètre important pour juger les performances d'une méthode d'élimination des ECG dans les EMG notamment les méthodes destinées à des applications nécessitant une réponse mécanique comme le contrôle de prothèses ou la ventilation mécanique.

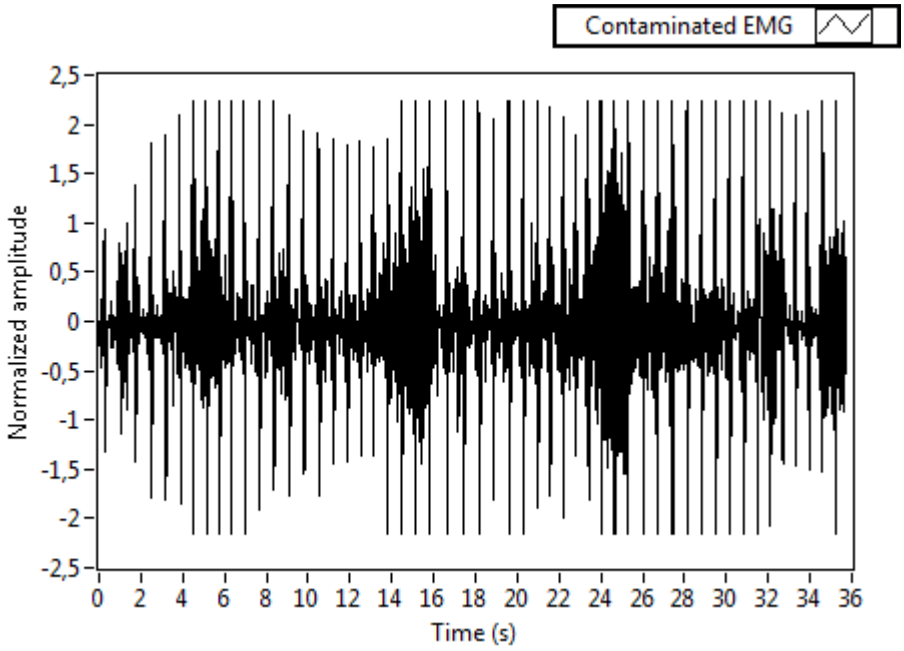


Figure 2.7 : Signal EMGdi brut

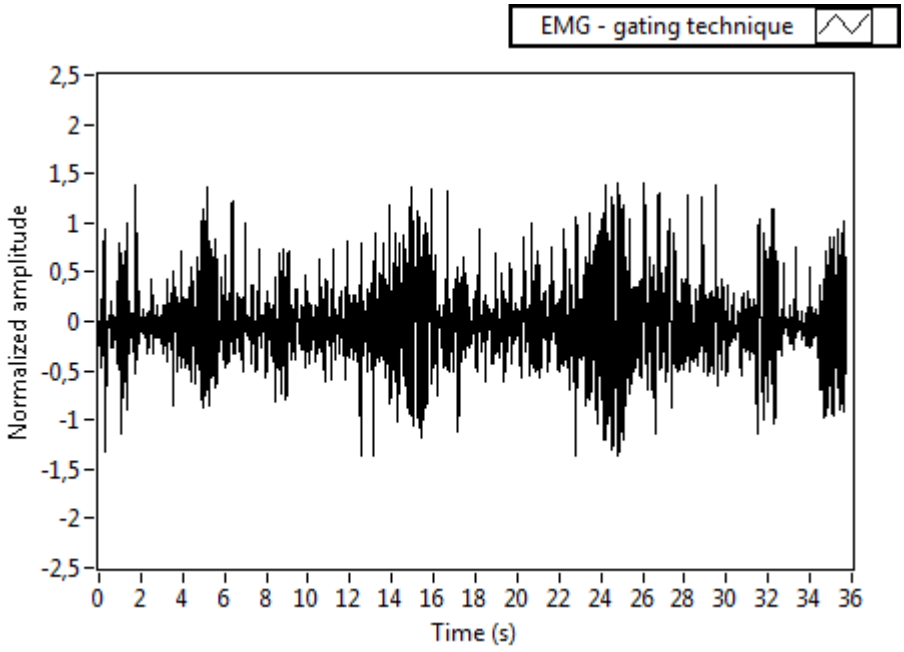


Figure 2.8 : EMGdi brut après traitement par la technique de seuillage en utilisant un seuil normalisé optimisé égal à 1.4

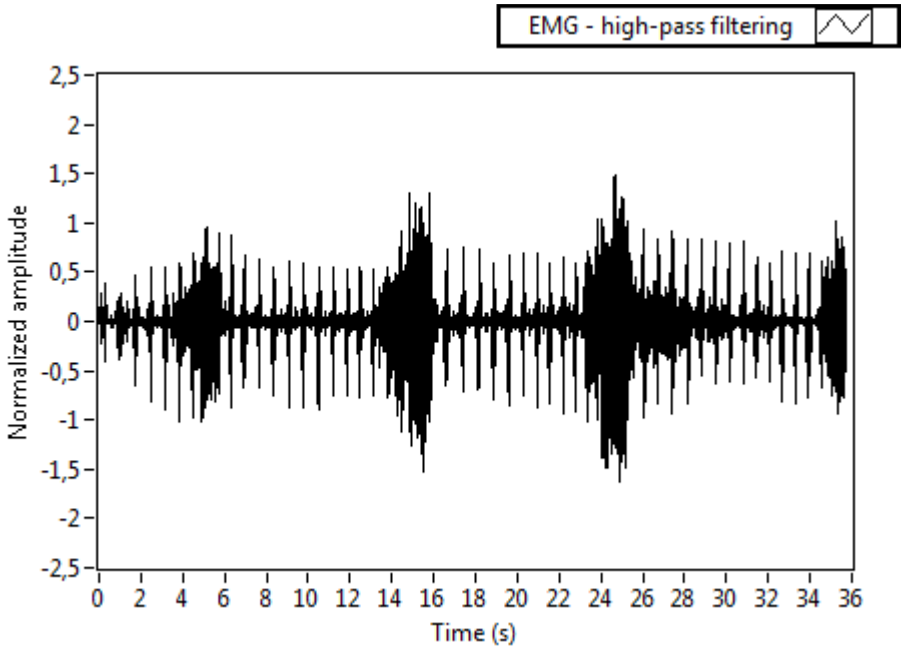


Figure 2.9 : Signal EMGdi brut après filtrage par un filtre passe-haut de Butterworth en utilisant une fréquence de coupure égale à 30 Hz et un ordre égal à 4

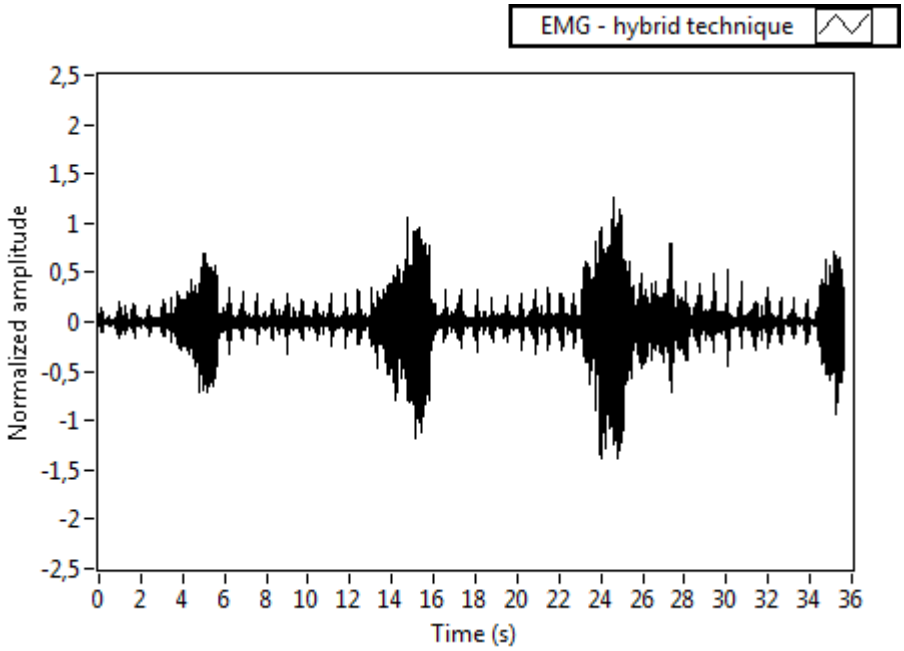


Figure 2.10 : Signal EMGdi brut après l'application de la technique hybride en utilisant les paramètres suivants : fc = 30 Hz, ordre = 4, l = 100 ms, g = 2

Tableau 2.2: Comparaison des méthodes retenues

	EMG brut	**Technique hybride**	**Filtrage passe-haut**	**Technique de coupage**
Vitesse	----------	Moyenne	Élevée	Faible
ARV(ECG)	0.269	0.047	0.073	0.134
ARV(EMG)	0.40	0.34	0.39	0.27
SNR	1	4.86	3.59	1.35

Le rapport signal sur bruit utilisé dans cette évaluation a été calculé selon la formule suivante [Zhou *et al.*, 2007]:

$$SNR(M)=\frac{ARV(M\_EMG)\times ARV(clean\_ECG)}{ARV(M\_ECG)\times ARV(clean\_EMG)} \tag{2.8}$$

(M) représente la méthode pour laquelle le paramètre SNR est calculé.

ARV(M_EMG) et ARV(M_ECG) sont les paramètres ARV calculés pour des portions d'EMG et d'ECG dans le signal EMG filtré par la méthode représentée par (M).

La moyenne ARV est calculée, pour un signal s donné, par l'équation suivante:

$$ARV=\frac{1}{N}\sum_{k=1}^{N}|s(k)| \tag{2.9}$$

Étant donné qu'on ne dispose pas d'un canal pour avoir les ECG pures et les EMG pures, ARV(clean_ECG) a été calculé en prenant les signaux ECG pendant les périodes expiratoires dans

lesquelles le signal EMG est quasiment absent. D'autre part, entre deux pics ECG, la contamination ECG est également quasiment absente. Ainsi, on a choisi de calculer la moyenne ARV(M_EMG) entre deux pics ECG étant donné qu'on ne peut disposer en pratique d'un signal EMG non contaminé. Ces deux paramètres nous permettent d'avoir une information importante sur les atténuations de la contamination ECG et de l'information EMG. Ces deux paramètres sont utilisés pour calculer le rapport SNR qui donne un indicateur global sur l'atténuation du signal EMG et de la contamination ECG.

Les résultats exprimés dans le tableau 2.2 montrent que la technique hybride a le SNR le plus élevé. Ainsi, cette technique est la plus performante pour des applications de contrôle mécanique tel que le contrôle de prothèses myoélectriques ou de ventilation mécanique. D'autre part, la vitesse de traitement du filtrage passe-haut est naturellement plus élevée que celle de la technique hybride. Cependant, la vitesse de traitement de la technique hybride reste convenable pour des applications temps-réel.

Concernant la valeur ARV, on constate que la valeur ARV(ECG) de la technique hybride est inférieure à celle des deux autres méthodes ce qui traduit une élimination plus importante des ECG dans la technique hybride que dans les autres méthodes. D'autre part, la valeur ARV(EMG) de la technique hybride et du filtrage passe-haut sont proches de la valeur ARV(EMG) du signal brut, ce qui montre qu'il n'y a pas de pertes considérables dans le signal EMG dans ces deux méthodes. Cependant, la valeur ARV(EMG) pour la technique de seuillage montre qu'il y a une perte de 32.5% des données EMG, ce qui est largement supérieur aux pertes dans les deux autres méthodes.

Ces résultats nous montrent que la technique hybride est la technique la plus performante en ce qui concerne l'élimination de la contamination ECG tout en permettant de conserver l'information EMG d'une manière similaire au filtrage passe-haut. Concernant la vitesse de traitement, la technique hybride a théoriquement une vitesse lui permettant d'opérer en temps-réel même si cette vitesse est légèrement inférieure à celle du filtrage passe-haut. Ainsi, globalement, la technique hybride est la technique qui serait la plus convenable pour l'élimination des ECG dans les EMG pour des applications temps-réel. Cependant, cette technique a uniquement été testée en simulation dans [Zhou *et al.*, 2007]. On a ainsi implémenté cette technique matérielle pour la première fois dans [Rhou *et al.*, 2008] sur un système à microcontrôleurs. Cette implémentation, ainsi qu'une architecture VHDL implémentant cette technique seront présentées dans le chapitre suivant. On proposera également des améliorations touchant la vitesse de traitement de cette technique ainsi que ses performances et on proposera une nouvelle méthode basée sur la technique hybride permettant d'avoir de meilleurs résultats tout en opérant plus rapidement et permettant également de détecter automatiquement et en temps réel l'activité musculaire du diaphragme.

2.3 Comparaison des méthodes de détection de l'activité musculaire

Comme présenté précédemment dans le premier chapitre, on peut distinguer trois catégories principales de méthodes de détection de l'activité musculaire en temps réel : les méthodes de seuillage classiques, la méthode basée sur l'algorithme itératif de détection de la respiration et

les méthodes de seuillage avancées. Le tableau 2.3 résume d'une manière générale les caractéristiques des méthodes de détection abordées.

Liste des caractéristiques des méthodes comparées:

1 : Détection d'une grande partie des EMG.

2 : Faible taux de fausses détections.

3 : Technique simple à mettre en œuvre.

4 : Possibilité d'utilisation dans une application temps réel.

5 : Technique pouvant opérer d'une manière autonome.

6 : Ne nécessite pas un canal supplémentaire pour l'acquisition des ECG.

Tableau 2.3: Caractéristiques des méthodes de détection de l'activité musculaire

Méthode	**Caractéristiques**					
	1	**2**	**3**	**4**	**5**	**6**
Méthodes de seuillage basiques			X	X	X	X
Algorithme itératif	X			X	X	X
Méthodes de seuillages avancées	X	X		X	X	X

Comme on peut le voir dans le tableau 2.3, les méthodes de seuillage avancées sont les méthodes qui satisfont le plus de critères exigés dans le cadre de notre application. Cependant,

les méthodes de détection de l'activité musculaire sont généralement appliquées sur des signaux EMGdi après leur traitement pour l'élimination du bruit et de la contamination ECG. Ainsi, leur efficacité ne peut être validée que pour les signaux sur lesquels ils ont été testés et sont, par conséquent, dépendant également de la méthode du traitement de l'EMG utilisée comme étape de prétraitement ainsi que de la méthode d'acquisition des signaux EMG qui détermine le degré de contamination des signaux EMG. Ainsi, il est nécessaire d'adapter toute méthode de détection choisie avec le type des signaux EMG utilisés et leur méthode de traitement.

2.4 Conclusion

Dans ce chapitre, nous avons présenté une étude comparative des méthodes d'élimination de la contamination ECG dans les signaux EMG. Nous nous sommes basés dans cette étude sur les caractéristiques de ces méthodes et leur compatibilité avec notre application. Nous avons également effectué une comparaison logicielle en utilisant des signaux EMGdi réels acquis via notre système d'acquisition des signaux respiratoires afin de comparer les performances des méthodes qu'on avait choisi dans une première étape de notre étude comparative. À travers cette étude comparative, on a conclu que la technique hybride est la technique la mieux adaptée à notre application car elle offre les meilleures performances tout en pouvant opérer d'une manière autonome en temps-réel. Cependant, cette technique n'avait jamais été testée matériellement et certaines améliorations pourraient être apportées à la vitesse de traitement et à la capacité d'élimination de la contamination ECG.

Ce chapitre a également présenté, d'une manière plus brève, une comparaison des méthodes de détection de l'activité musculaire basée sur l'EMG présentés dans le premier chapitre. Cette comparaison nous a montré que les méthodes de seuillages avancées sont les méthodes les plus convenables pour notre application.

Dans le chapitre qui suit, nous proposerons des améliorations de la technique hybride et nous validerons son fonctionnement matériel. On proposera finalement une méthode complète permettant d'éliminer la contamination ECG tout en permettant de détecter l'activité musculaire. Cette méthode sera basée essentiellement sur la technique hybride améliorée et une méthode de seuillage avancée adaptée à notre application.

CHAPITRE 3 VALIDATION MATÉRIELLE, NOUVELLE ARCHITECTURE ET NOUVELLE MÉTHODE GLOBALE D'ÉLIMINATION DES ECG ET DE DÉTECTION DE L'ACTIVITÉ MUSCULAIRE EN TEMPS-RÉEL

3.1 Introduction

Étant donné que l'étude comparative, présentée dans le chapitre précédent, a démontré que la technique hybride était la mieux adaptée à notre application, nous avons opté pour cette méthode. Cependant, cette dernière a été examinée uniquement en simulation et malgré le fait que cette technique soit plus performante que les autres techniques présentées, quelques améliorations touchant l'élimination des ECG pendant les périodes expiratoires et la vitesse de traitement peuvent être apportées. Ainsi, dans une première partie, nous avons testé la technique hybride sur un système à microcontrôleurs afin de valider son fonctionnement matériel. Nous avons ensuite proposé une architecture VHDL implémentant l'algorithme SC, composant innovateur de la technique hybride présenté dans le chapitre précédent. D'un autre côté, dans une seconde partie, nous avons proposé une méthode globale d'élimination des ECG et de détection de l'activité musculaire du diaphragme en se basant sur la technique hybride améliorée et une méthode adaptée de seuillage avancé. Une implémentation de cette méthode globale en langage Labview sera proposée ainsi qu'un logiciel regroupant la technique de seuillage, le filtrage passe-haut, la technique hybride et la méthode globale proposée.

3.2 Validation matérielle de la technique hybride

3.2.1 Architecture du système

La technique hybride a été implémentée sur un système à microcontrôleurs (figure 3.1). Ce système est basé sur deux microcontrôleurs 8-bits de la famille AVR d'ATMEL (ATMEGA16 et ATMEGA8515) munis d'une mémoire SDRAM interne de 1 Ko et pouvant fonctionner à une fréquence d'opération réglable de 0 à 16 MHz. En plus de ces deux microcontrôleurs, le système est également composé de deux liens séries RS-232 et d'un composant MAX-232 pour adapter les signaux RS-232 (+/- 10V) à la logique TTL (0/5V).

Les microcontrôleurs ATMEGA16 et ATMEGA8515 ont été choisis pour cette implémentation car ils faciliteront l'intégration future des programmes réalisés dans le système d'acquisition des signaux respiratoires utilisant également deux microcontrôleurs du même type. Ce choix convient aussi aux exigences de l'implémentation de la technique hybride, notamment en ce qui concerne la taille de la mémoire interne, la fréquence de fonctionnement et les interfaces de communication. De plus, les microcontrôleurs utilisés ont une architecture matérielle Harvard et une architecture logicielle RISC, ce qui rend l'exécution de leurs programmes beaucoup plus rapide que les microcontrôleurs classiques utilisant une architecture matérielle Van Neumann et une architecture logicielle CISC. En effet, dans l'architecture Harvard, contrairement à l'architecture classique Van Neumann, les instructions et les données sont contenues dans des mémoires différentes et sont véhiculées dans des bus indépendants. Du côté de l'architecture logicielle, l'architecture RISC utilise une structure de type pipeline et utilise des instructions codés sur un mot, ce qui permet de lire le code d'une instruction en un seul cycle machine et

d'exécuter une instruction tout en recherchant l'instruction suivante en mémoire en un seul cycle machine [Tavernier, 2002].

Afin de tester notre système, deux interfaces de communication en Labview ont été réalisées. La première interface envoie les signaux EMGdi échantillonnées, préalablement enregistrés dans un fichier texte, à la première liaison série du système qui les envoie à son tour au MAX-233 pour les adapter à la logique TTL des microcontrôleurs. Ces signaux, une fois adaptés, sont envoyés par la suite au microcontrôleur maitre (ATMEGA16) qui implémente l'algorithme de la technique hybride en langage C afin d'éliminer la contamination ECG. Le microcontrôleur maitre envoie les signaux EMGdi traités par la technique hybride au microcontrôleur esclave (ATMEGA8515) via une liaison SPI. Ce dernier, programmé en langage assembleur, est chargé de gérer la transmission de ces signaux via la seconde liaison série RS-232 vers une seconde interface Labview qui affiche les résultats du traitement par le système.

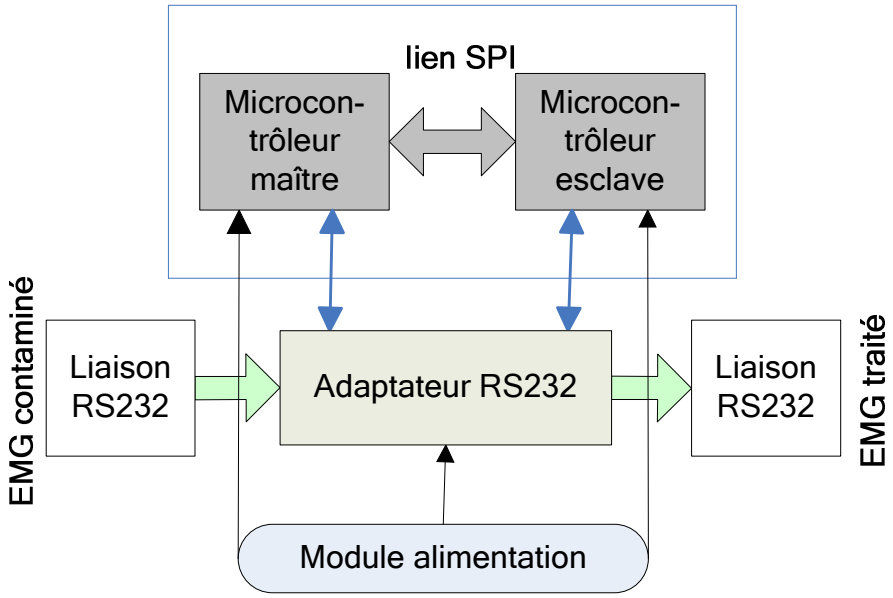


Figure 3.1 : Schéma bloc du système à deux microcontrôleurs

3.2.2 Algorithme implémenté

Le microcontrôleur ATMEGA16 joue le rôle principal dans le système réalisé. En effet, la méthode hybride est complètement implémentée dans ce dernier qui reçoit les mesures initiales et les envoie au microcontrôleur ATMEGA8515 servant d'interface entre le microcontrôleur maître ATMEGA15 et l'interface LabView de réception des résultats. Les ordinogrammes des programmes implémentés dans les microcontrôleurs ATMEGA16 et ATMEGA8515 sont présentés dans les figures 3.2, 3.3 et 3.4. Les programmes assembleur et C implémentés sont présentés dans l'annexe A.

L'algorithme hybride implémenté est décrit dans la section 2.2.2.1 avec une légère amélioration afin diminuer le nombre d'opérations effectuées par le microcontrôleur maître à chaque itération. En effet, l'équation 2.7, présentée dans la section 2.2.2.1 a été remplacée par l'équation suivante :

$$\mathrm{ARV(m)} = \mathrm{ARV(m-1)} + \frac{1}{\mathrm{N}}(|\mathrm{o(N+m-1)}| - |\mathrm{o(m-1)}|) \qquad (3.1)$$

L'équation 3.1 a été obtenue en décomposant l'équation 2.7 :

$$\mathrm{ARV(m)} = \frac{1}{\mathrm{N}}\sum_{k=m}^{N+m-1}|o(k)| = \frac{1}{\mathrm{N}}\sum_{k=m-1}^{N+m-2}|o(k)| - \frac{1}{\mathrm{N}}|\mathrm{o(m-1)}| + \frac{1}{\mathrm{N}}|\mathrm{o(N+m-1)}| \qquad (3.2)$$

Ce choix est justifié par le fait que l'équation (3.1) permet de diminuer les opérations nécessaires à chaque itération pour calculer la moyenne ARV de N+1 opérations à trois opérations, sachant que N est un paramètre ajustable de l'ordre de 200 en général. L'utilisation de cette équation permet aussi la réduction du délai traitement par le biais de la réduction de

l'accès à la mémoire de données contenant les éléments utilisés dans la somme de l'équation 2.7.

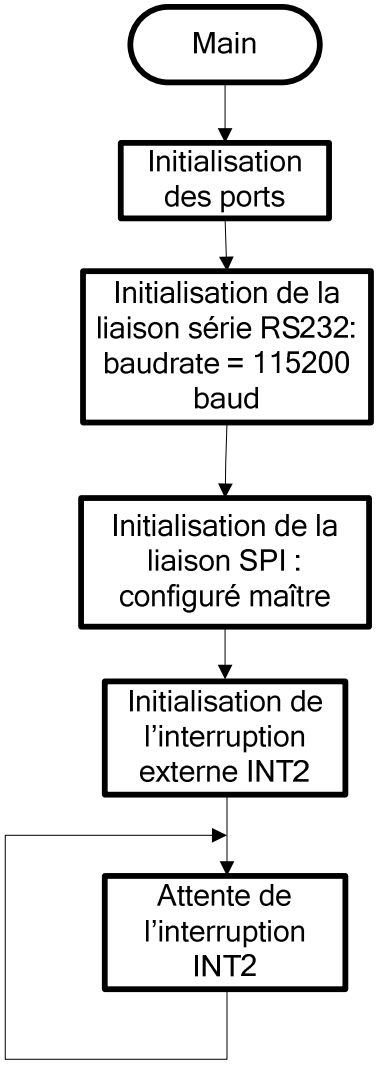


Figure 3.2 : Ordinogramme du programme principal de l'ATMEGA16

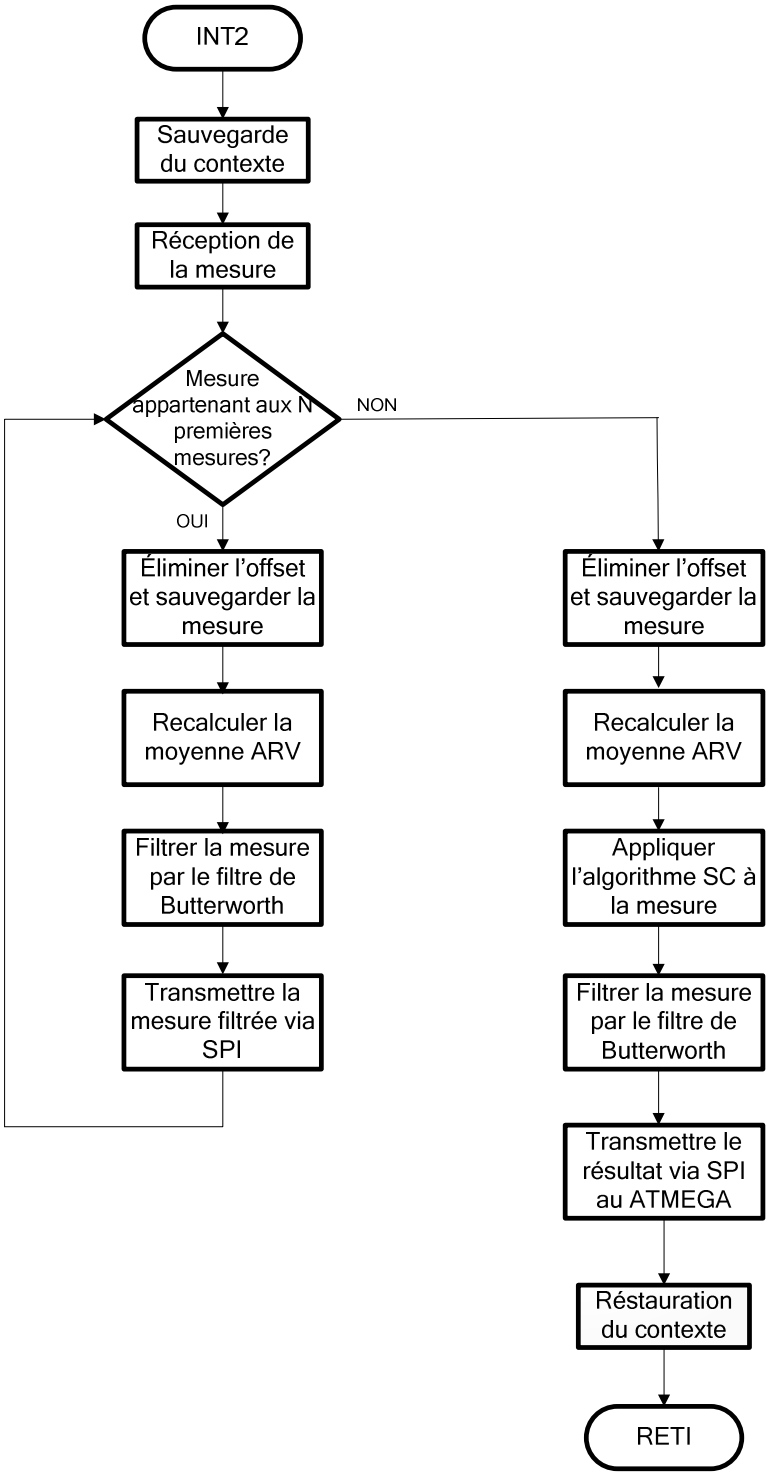


Figure 3.3 : Ordinogramme du programme d'interruption du microcontrôleur maître

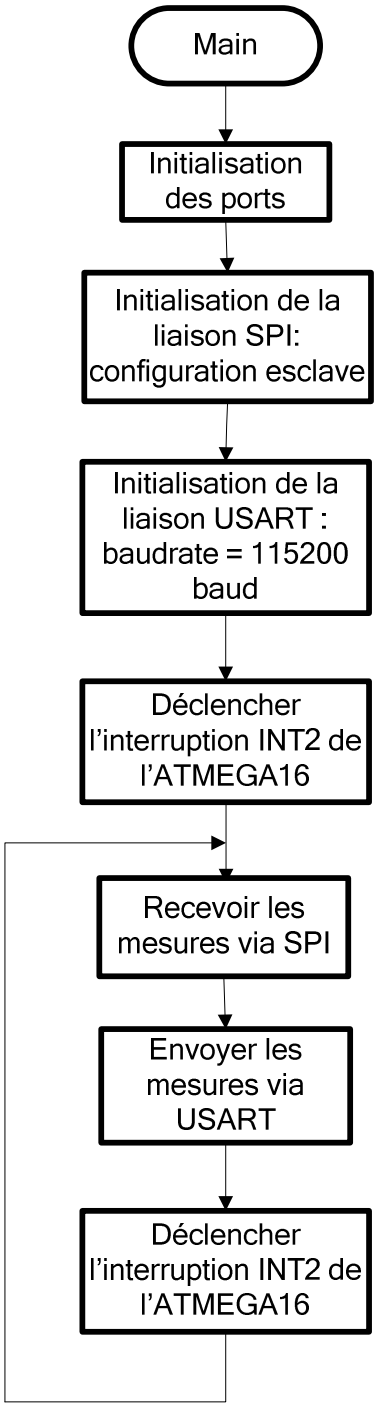


Figure 3.4 : Ordinogramme du programme principal de l'ATMEGA8515

3.3 Architecture VHDL de l'algorithme Spike-Clipping (SC)

3.3.1 Calcul de la moyenne ARV(1) et du premier échantillon EMG traité

3.3.1.1 Module de calcul de la moyenne ARV(1)

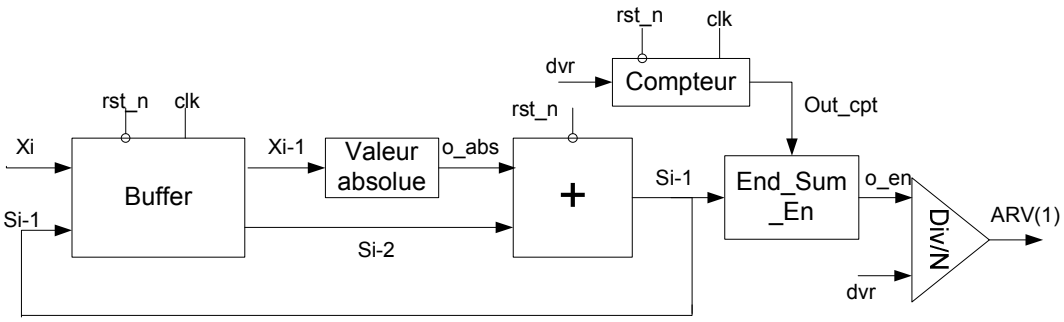


Figure 3.5 : Éléments du module de calcul de la moyenne ARV(1)

Ce module (figure 3.5) implémente le traitement initial qui doit être effectué dans l'algorithme SC afin de calculer la première moyenne ARV qui sera par la suite utilisée dans les étapes ultérieures de l'algorithme SC. À chaque coup d'horloge, le bloc « Buffer » envoie en sortie un échantillon xi de l'EMG ainsi que la dernière somme calculée. Le point EMG envoyé, constituant l'échantillon le plus récent, est par la suite additionné à la dernière somme calculée. Cette opération est répétée jusqu'à l'activation du bloc « End_Sum_En » par le bloc « Compteur » après N comptages. Par la suite, le bloc « End_Sum_En » envoie en sortie la somme des N premiers échantillons EMG. Cette somme est divisée par le bloc « Div/N » par le nombre N. Ainsi, le résultat en sortie du bloc « Div/N » sera la moyenne ARV(1). L'implémentation du diviseur utilisée est expliquée en détail dans l'annexe B.

3.3.1.2 Module de sélection de la première sortie de l'algorithme SC

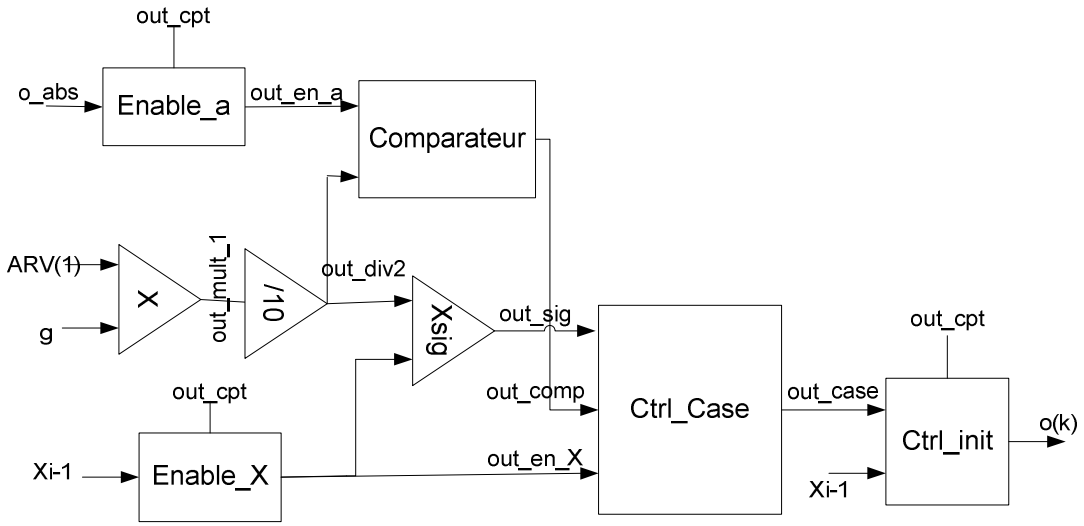


Figure 3.6 : Éléments du module de sélection de la première sortie de l'algorithme SC

Le module présenté dans la figure 3.6 permet de calculer la première sortie de l'algorithme SC en se basant sur les valeurs de o_abs, ARV(1) et Xi-1 obtenues par le module de calcul de la moyenne ARV(1). Le module Enable_a, activé par l'entrée out_cpt, permet l'envoi de la valeur absolue de l'entrée courante à la fin du calcul de la moyenne ARV(1). Cette dernière est comparée à la valeur de g × ARV(1) par le bloc Comparateur qui envoie par la suite une valeur binaire dépendamment du résultat de la comparaison au bloc Ctrl_Case. Le module Ctrl_Case envoie en sortie la valeur de l'entrée X(N+1) (lorsque le bloc Enable_X est activé après la fin du calcul de ARV(1)) ou de l'entrée signe(x(N+ 1)) × g × ARV(1) selon de résultat de comparaison envoyé par le bloc Comparateur. Le bloc Ctrl_init est utilisé en sortie de ce module afin de permettre de mettre en sortie les N premières entrées qui restent inchangées pendant le calcul de la moyenne ARV(1) et de mettre en sortie le résultat obtenu par le bloc Ctrl_case une fois le calcul de la moyenne ARV(1) achevé.

3.3.2 Module de calcul de la moyenne ARV(m)

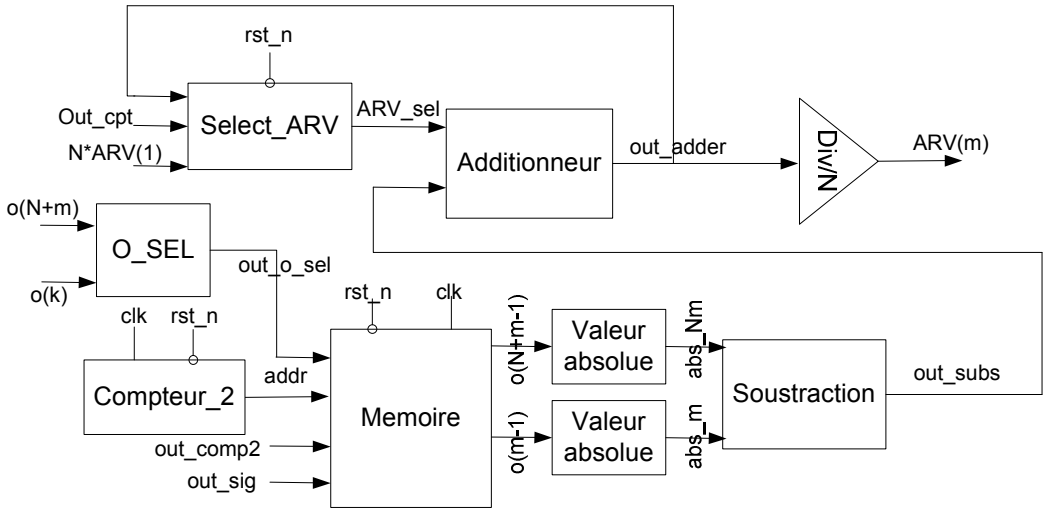


Figure 3.7 : Éléments du module de calcul de la moyenne ARV(N)

Ce module (figure 3.7) permet de calculer la moyenne ARV(m), avec m > 1, en se basant sur l'équation 3.1. L'utilisation de cette équation au lieu de l'équation classique de la moyenne ARV décrite dans 2.7 permettra, comme on l'a expliqué dans la section 3.2.2, de réduire le temps de traitement de l'algorithme SC.

Le module mémoire permet de stocker les sorties précédentes de l'algorithme afin de calculer la moyenne ARV. Le bloc Compteur_2, synchronisé avec l'envoi des points EMG en entrée, permet d'incrémenter l'adresse de stockage de la sortie courante. Les points stockés sont utilisés par la suite pour calculer la nouvelle moyenne ARV.

3.3.3 Module comparateur

À cette étape de l'algorithme SC, le bloc Comparateur_2 (figure 3.8) compare les valeurs de out_abs2 et out_mult2 représentant respectivement les valeurs de $|x(N+m)|$ et $g \times ARV(m)$: si

out_abs2 > out_mult2, alors le bloc Comparateur_2 envoie en sortie la valeur de out_mult2. Dans le cas contraire, le bloc Comparateur_2 envoie en sortie la valeur de out_abs2.

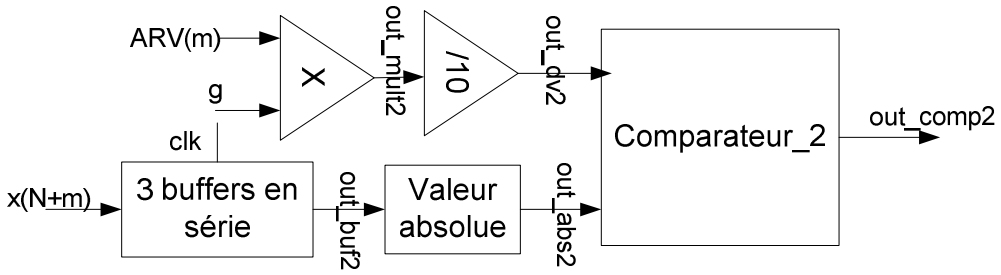


Figure 3.8 : Éléments du module comparateur

3.3.4 Module de sélection des points de sortie de l'algorithme

Ce module, décrit dans la figure 3.9, permet d'implémenter l'opération suivante : si out_comp2 est activé, alors :

$$o(N+m) = signe(x(N+m)) \times out\_mult2 \quad (3.3)$$

sinon :

$$o(N+m) = out\_buf2 \quad (3.4)$$

Ainsi, ce module permet de fournir les sorties o(N+m) présentant les points EMG traités par l'algorithme SC.

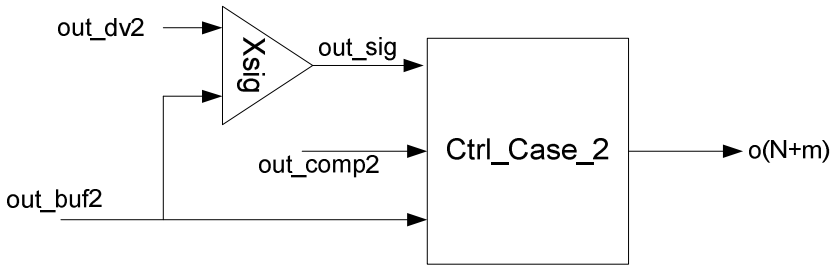


Figure 3.9 : Éléments du module de sélection des points de sortie de l'algorithme SC

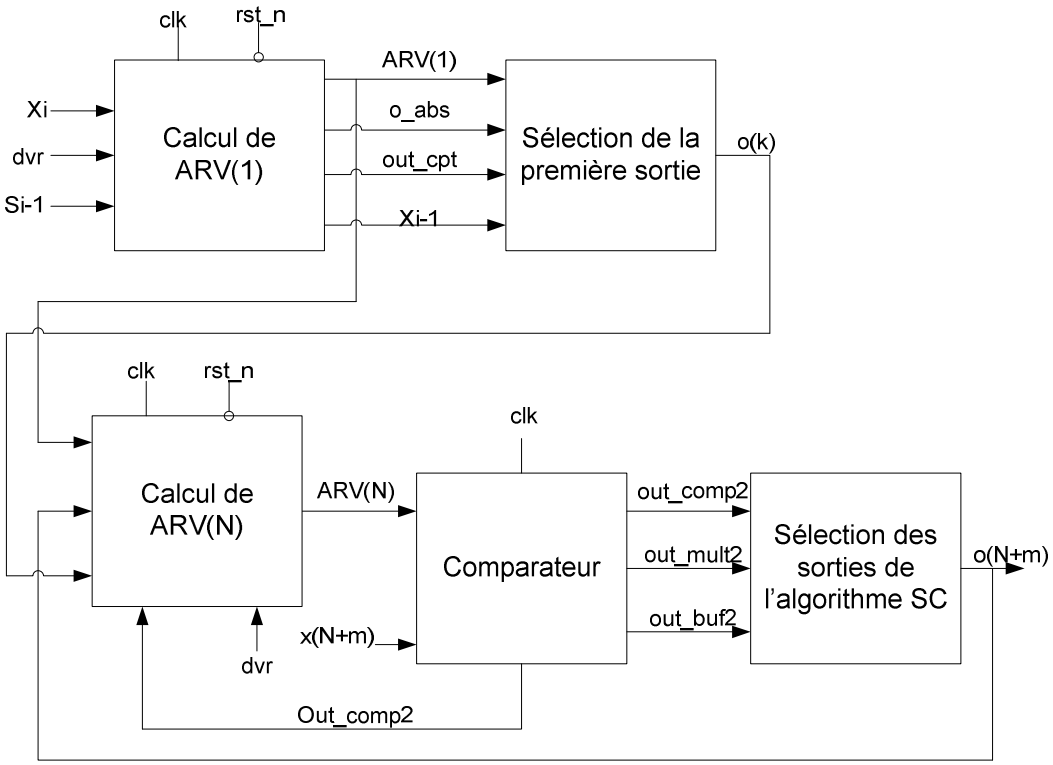


Figure 3.10 : Schéma bloc global de l'implémentation de l'algorithme SC

3.3.5 Relation entre les différents modules de l'architecture implémentée

La figure 3.10 présente l'architecture globale décrivant la relation entre les différents modules implémentant l'algorithme SC. Cette architecture a été implémentée en langage VHDL. Les résultats expérimentaux seront présentés dans le prochain chapitre.

3.4 Nouvelle méthode globale

3.4.1 Description de la méthode

La nouvelle méthode que nous proposons combine l'élimination efficace de la contamination ECG et la détection de l'activité musculaire en temps-réel.

L'algorithme de notre méthode est composé de deux parties principales : une première partie implémentant la technique hybride utilisant la formule de calcul accéléré de la moyenne ARV, et une seconde partie traitant les résultats obtenus afin d'obtenir l'activation/désactivation de l'activité musculaire et le signal EMG final après l'élimination de la contamination. L'algorithme global de la méthode proposée est comme suit :

1- La première étape consiste à évaluer la valeur des paramètres utilisés, soit le gain (g), la longueur de fenêtre (l), la fréquence de coupure fc, l'ordre du filtre passe-haut, et la fenêtre fixée (l'). Les valeurs de ces paramètres sont choisies par l'utilisateur dépendamment des performances recherchées. Dans le cadre de notre application, les valeurs du gain, de la longueur de la fenêtre, de la fréquence de coupure et de l'ordre du filtre ont été choisies comme suit : g = 2, l = 100 ms, fc = 30 Hz, ordre du filtre = 4. Concernant la fenêtre fixée (l'),

elle est exprimée par la formule: $l' = N' \times Te$, où Te est la période d'échantillonnage des signaux EMG bruts, et N' le nombre de points échantillonnés se trouvant la fenêtre l'. Dans le cadre de notre application, l' sera fixée à 850 ms.

2- Initialement, la moyenne ARV est calculée en utilisant la relation :

$$ARV(1) = \frac{1}{N}\sum_{k=1}^{N} |o(k)| = \frac{1}{N}\sum_{k=1}^{N} |x(k)| \quad (3.5)$$

où $N = l \times fe$, x le signal d'entrée du bloc SC et o le signal de sortie de ce bloc. À cette étape de l'algorithme, les sorties o et les entrées x sont égales.

3- À l'instant N+1, si $|x(N+1)| > g \times ARV(1)$, alors on pose :

$$o(N+1) = sgn(x(N+1)) \times g \times ARV(1) \quad (3.6)$$

où sgn est la fonction signe.

Si $|x(N+1)| \leq g \times ARV(1)$ alors on pose :

$$o(N+1) = x(N+1) \quad (3.7)$$

4- On recalcule la moyenne ARV(2) en utilisant la formule de calcul accéléré : $ARV(2) = ARV(1) + \frac{1}{N}(|o(N+1)| - |o(1)|)$ (3.8)

5- On refait ce processus pour tous les points du signal EMG brut. Ainsi, pour l'instant d'échantillonnage $N + m$, où m > 1, on calcule la moyenne AVR en utilisant la formule :

$$ARV(m) = ARV(m-1) + \frac{1}{N}(|o(N+m-1)| - |o(m-1)|) \quad (3.9)$$

Si $|x(N+m)| > g \times ARV(m)$, alors on pose :

$$o(N+m) = sgn(x(N+m)) \times g \times ARV(m) \quad (3.10)$$

Sinon, on pose :

$o(N+m) = x(N+ m)$ (3.11)

6- Pendant le calcul de la moyenne ARV(1), les points de sorties o(n) où n ≤ N, sont égaux aux points d'entrée x(n). Les points de sorties o(n) du bloc SC passent par la suite par un filtre numérique passe-haut de Butterworth du quatrième ordre avec une fréquence de coupure de 30Hz. La sortie y de ce filtre est définie par l'équation suivante :

$$\begin{aligned} &y(n) = \\ &o(n-4) - 4\times o(n-3) + 6\times o(n-2) - 4\times o(n-1) \\ &+o(n) - 0.7816187403\times y(n-4) + 3.3189386048\times y(n-3) \\ &-5.2911525842\times y(n-2) + 3.7537627567\times y(n-1) \end{aligned} \quad (3.12)$$

À partir de l'instant N + 1, les points de sorties o(N+m) sont filtrés en utilisant le filtre défini par l'équation suivante:

$$\begin{aligned} &y(N+m) = \\ &o(N+m-4) - 4\times o(N+m-3) + 6\times o(N+m-2) \\ &-4\times o(N+m-1) + o(N+M) - 0.7816187403\times y(N+m-4) \\ &+3.3189386048\times y(N+m-3) - 5.2911525842\times y(N+m-2) \\ &+3.7537627567\times y(N+m-1) \end{aligned} \quad (3.13)$$

7- Les points y sont utilisés pour calculer une nouvelle moyenne en utilisant la formule:

$$ARV'(m') = \frac{1}{N'}\sum_{k=m'}^{N'+m'-1}|y(k)| \quad (3.14)$$

où $l' = N'\times Te$ et $m' = N+m-N'$ lorsque N + m > N'.

Un seuil s devrait être fixé afin de l'utiliser dans la comparaison de la prochaine étape.

Afin d'accélérer les calculs, la moyenne ARV' peut être calculée en utilisant la relation :

$$ARV'(m') = ARV'(m'-1) + \frac{1}{N}(|y(N'+m'-1)| - |y(m'-1)|) \quad (3.15)$$

8- À l'instant N'+m'=N+m, lorsque N + m > N' :

   Si ARV'(m') < s, l'algorithme considère que le point y(N+m) est un point du signal à un instant où il n'y a aucune activité musculaire (dans le cas des signaux EMGdi, ceci correspond à une période expiratoire). Dans ce cas, la détection de l'activité musculaire et la sortie z sont mises à 0, où z représente le signal EMG final après l'élimination de la contamination ECG. Dans le cas contraire, la détection de l'activité musculaire est mise à 1 et z(N+m) = y(N+m). Ceci permet de détecter en temps réel toute activité musculaire tout en éliminant la contamination ECG restante dans les signaux EMG pendant l'absence de l'activité musculaire.

Le schéma bloc de cette méthode globale proposée pour l'élimination de la contamination ECG accélérée tout en détectant simultanément l'activité musculaire est présenté dans la figure 3.11.

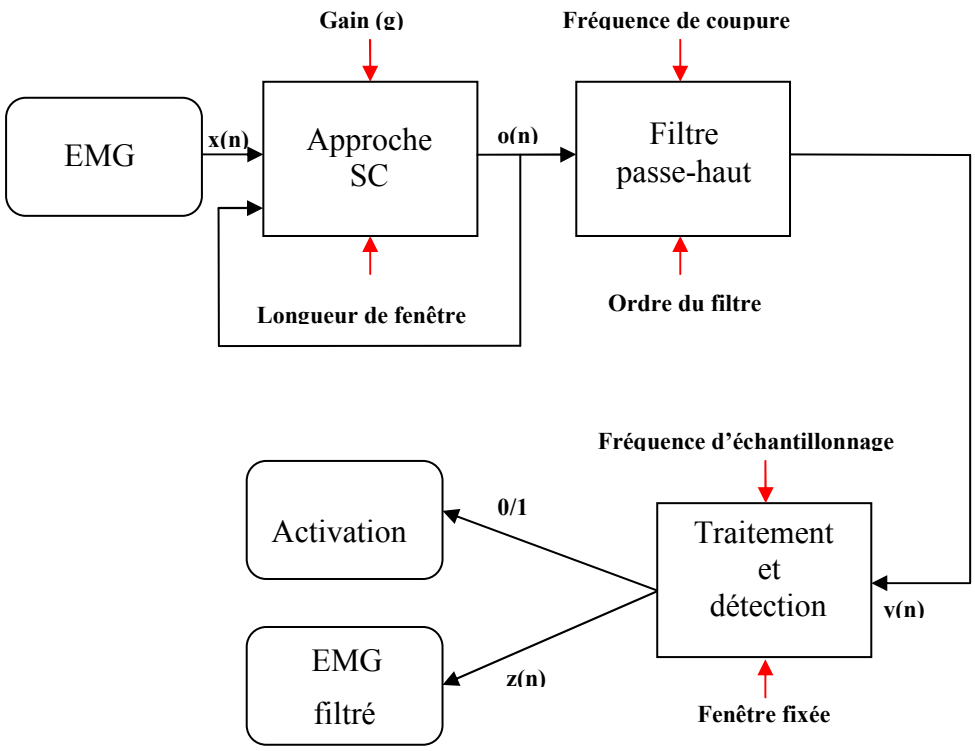


Figure 3.11 : Schéma bloc de la méthode globale proposée d'élimination de la contamination ECG et de la détection de l'activité musculaire

3.4.2 Avantages de la méthode

La méthode qu'on propose offre des performances supérieures par rapport à la technique hybride dans la mesure où elle offre la possibilité d'éliminer complètement les signaux ECG de faibles amplitudes ainsi que les bruits pendant les périodes de relaxation musculaire, ce qui diminue le risque de fausses interprétations. Ajouté à cela, notre méthode permet une détection quasi-instantanée de l'activation et la désactivation musculaire, car, contrairement aux méthodes classiques présentées dans le premier chapitre qui impliquent un délai d'attente

avant de confirmer une détection, notre méthode n'implique pas de délais car elle utilise un calcul de moyenne sur une fenêtre beaucoup plus large que celles utilisées dans les autres méthodes, ce qui augmente considérablement le degré de confiance en l'exactitude d'une détection et permet de valider une détection sans avoir besoin d'informations supplémentaires collectées pendant les délais d'attente.

D'un autre côté, notre méthode offre l'avantage de combiner les opérations de filtrage et de détection de l'activité musculaire en temps-réel et d'une manière automatique dans un seul système.

Cette méthode diminue également d'une manière considérable le temps de traitement pour les opérations de filtrage et de détection puisqu'elle utilise une partie de calcul commune à la détection de l'activité musculaire et à l'élimination de la contamination ECG et qu'elle utilise l'équation de calcul accéléré de la moyenne ARV (équation 3.1).

3.5 Logiciel de traitement des signaux EMG par diverses méthodes

Afin de valider le fonctionnement logiciel de la nouvelle méthode qu'on a proposé dans la section 3.4, et afin de regrouper la technique de coupage, le filtrage passe-haut, la technique hybride et la nouvelle méthode qu'on a proposé dans un seul logiciel, on a réalisé un logiciel en langage LabView regroupant les méthodes précédentes et permettant de contrôler, par le biais d'une interface utilisateur (figure 3.12), les paramètres de ces méthodes et la source des données. Ce logiciel est composé de cinq onglets. Le premier onglet correspondant au menu qui permet d'afficher le signal EMG initial et de choisir la méthode désirée ainsi que la source

d'extraction du signal. Les quatre autres onglets, correspondant à la technique de seuillage, le filtrage passe-haut, la technique hybride et la nouvelle méthode globale que nous avons proposée, permettent de fixer les paramètres correspondants à chaque méthode et d'afficher la courbe EMG traitée. Tous ces onglets ainsi que le programme Labview de ce logiciel sont présentés dans l'annexe C.

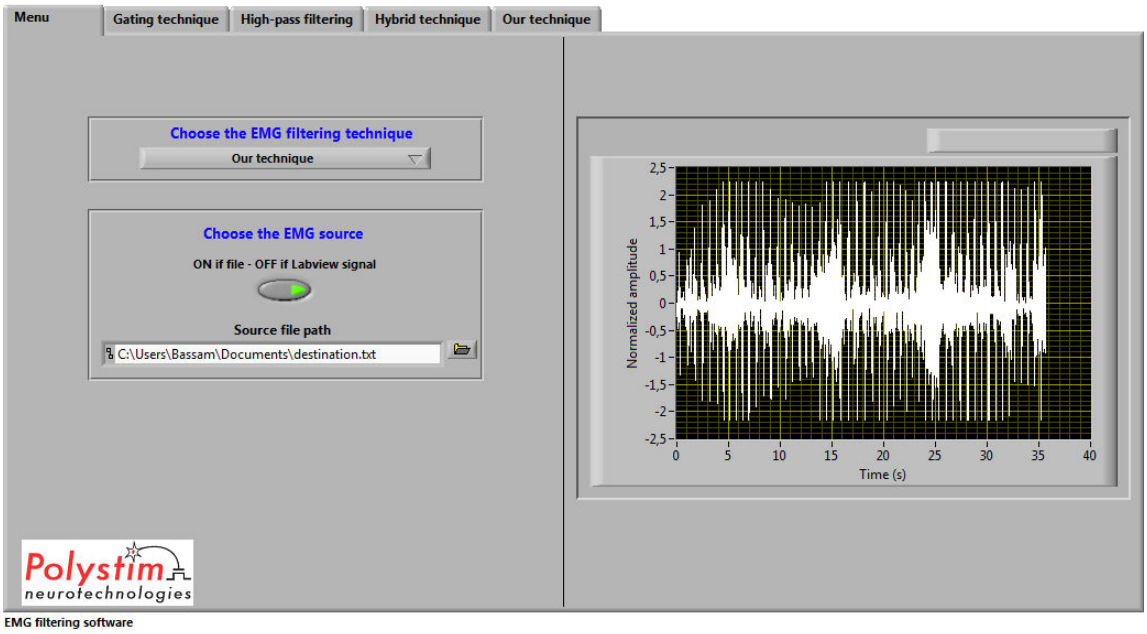


Figure 3.12 : Onglet principal du logiciel réalisé

3.6 Conclusion

Dans ce chapitre, nous avons proposé une implémentation de la technique hybride dans un système à microcontrôleurs. Cette implémentation a permis de démontrer, pour la première fois, le fonctionnement de la technique hybride sur une plate-forme matérielle.

Dans un second temps, nous avons proposé une nouvelle architecture VHDL implémentant l'algorithme SC qui constitue le composant novateur de la technique hybride.

Par la suite, nous avons proposé une nouvelle méthode, basée sur la technique hybride, permettant d'éliminer, d'une manière complète, la plupart des restes de la contamination ECG durant les périodes expiratoires et de détecter simultanément l'activité musculaire en général, et diaphragmatique en particulier. Cette méthode pourra être utilisée dans diverses applications biomédicales.

Finalement, nous avons présenté un logiciel LabView, réalisé dans le cadre de cette maîtrise, regroupant les techniques classiques d'élimination des ECG dans les EMG en temps réel ainsi que la technique hybride et la nouvelle méthode que nous avons proposé dans ce chapitre. Ce logiciel, en plus de permettre de valider la nouvelle méthode, permettra de faire des expérimentations sur les signaux EMG en sélectionnant simplement une méthode et les paramètres d'entrée de cette méthode.

Dans le chapitre qui suit, nous présenterons les résultats expérimentaux que nous avons obtenu pour les diverses contributions présentées dans ce chapitre.

CHAPITRE 4 RÉSULTATS

4.1 Introduction

Dans ce chapitre, on présentera les résultats expérimentaux obtenus pour le système à microcontrôleurs implémentant la méthode hybride ainsi que le logiciel LabView et l'architecture VHDL présentés dans le chapitre précédent. On utilisera pour nos expérimentations des signaux EMGdi réels acquis via le système d'acquisition des signaux respiratoires présenté dans le premier chapitre de ce mémoire. Pour des soucis de clarté, la technique hybride proposée dans la section 2.2.2.1 et que nous avons amélioré dans la section 3.2.2 sera notée MH et la nouvelle technique hybride que nous avons proposé dans la section 3.4.1 sera notée NTH.

4.2 Signaux EMG utilisés

Les EMG utilisés dans le cadre de nos expérimentations sont des EMG de type diaphragmatique pris sur une personne sans problèmes respiratoires via le système d'acquisition présenté dans le premier chapitre (figure 4.1).

Les étapes d'acquisition des signaux EMGdi peuvent être résumées comme suit : tout d'abord, un cathéter œsophagien de 2 mm a été inséré à travers l'œsophage en direction de l'estomac. Ce cathéter contient cinq électrodes-anneaux en platine séparés avec une distance 1.5 cm de largeur 2 mm permettant l'acquisition des EMGdi.

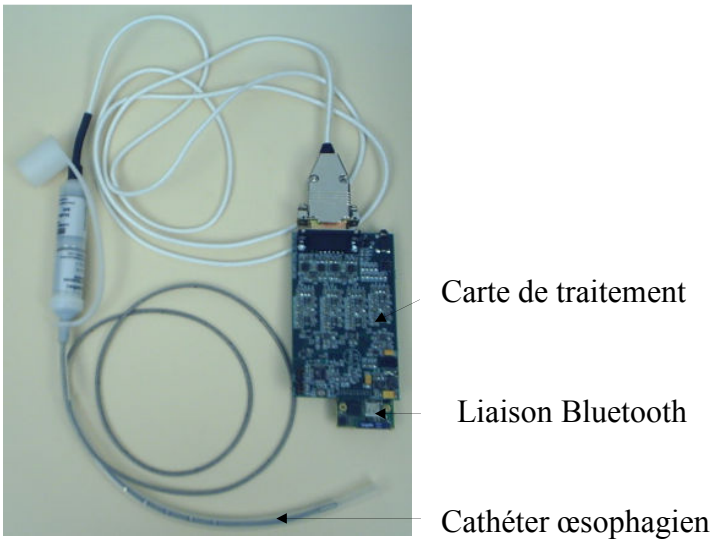


Figure 4.1 : Photographie du système d'acquisition sans fil [Rhou *et al.*, 2008]

Les signaux EMGdi de faible amplitude acquis via les électrodes-anneaux sont tout d'abord amplifiés et conditionnés par une carte de traitement externe. Une fois ces signaux amplifiés, ces derniers sont numérisés par le biais d'une CAN 16 bits avant d'être traités par deux microcontrôleurs de la famille AVR qui forment des trames. Ces trames sont par la suite envoyées à travers une liaison Bluetooth à un ordinateur afin de les enregistrer. La fréquence d'échantillonnage utilisée dans le cadre de notre expérimentation est de 2 kHz et le nombre d'échantillons utilisés est de 71350, ce qui est équivalent à un enregistrement d'une durée de 36 secondes environ. L'enregistrement EMGdi utilisé dans le cadre de nos expériences contient des blocs EMGdi correspondant à des inspirations fortes, faibles et normales ainsi à des inspirations de différentes durées. La fréquence d'échantillonnage utilisée pour nos signaux EMGdi a été fixée à la valeur 2 kHz afin d'obtenir une bonne résolution.

4.3 Vérification du fonctionnement du système à microcontrôleurs

4.3.1 Dispositif de test

Afin de tester le système à microcontrôleurs présenté dans le chapitre précédent, deux interfaces en langage LabView ont été réalisées pour l'envoi et la réception des échantillons EMGdi. Ainsi, une première interface (annexe C) récupère les signaux EMGdi enregistrés dans un fichier texte et les envoie via la liaison série RS-232 au microcontrôleur maître qui effectue le traitement par la méthode hybride avant d'envoyer les signaux traités au microcontrôleur esclave qui se charge de l'envoi des résultats à la seconde interface. La seconde interface (annexe C) se charge de récupérer les données traitées sur la liaison série RS-232 de l'ordinateur et de les afficher. Les deux interfaces permettent de choisir la vitesse en bauds désirée ainsi que le port de l'ordinateur sur lesquels les données peuvent être envoyées/reçues et affichent les signaux envoyés et reçus.

La vitesse de transmission/réception utilisée dans le cadre de cette expérience est de 115200 bauds pour les microcontrôleurs et les interfaces de communication, ce qui correspond à la vitesse maximum en utilisant les horloges internes des microcontrôleurs utilisés.

4.3.2 Paramètre à évaluer

Afin de montrer l'exactitude des résultats expérimentaux par rapport aux résultats obtenus en simulation (chapitre 2), on comparera les signaux EMGdi obtenus par la simulation LabView et les signaux EMGdi obtenus par le système à microcontrôleurs. On comparera également l'indice

ARV (Average Rectified Value) et le rapport signal/bruit des signaux EMGdi simulés et expérimentaux. Ces deux indices seront calculés en utilisant les deux équations présentées et expliquées dans la section 2.2.3:

$$SNR(M)=\frac{ARV(M\_EMG)\times ARV(clean\_ECG)}{ARV(M\_ECG)\times ARV(clean\_EMG)} \tag{4.1}$$

Et :

$$ARV=\frac{1}{N}\sum_{k=1}^{N}|s(k)| \tag{4.2}$$

Où s est le signal d'entrée EMG et N est le nombre d'échantillons du signal.

4.3.3 Résultats

La figure 4.2 représente les signaux EMGdi brut, EMGdi simulé et EMGdi expérimental. On peut constater que le signal EMGdi simulé et le signal EMGdi expérimental sont identiques.

Le tableau 4.1 montre que l'indice ARV, caractérisant la contamination ECG et le signal EMG, et le rapport signal sur bruit sont identiques pour le signal EMGdi simulé et le signal EMGdi expérimental. Ceci s'explique par le fait que le nombre de chiffres après la virgule dans les valeurs des échantillons EMGdi est limité à deux chiffres. Ainsi, étant donné que les opérations de traitement effectuées par la simulation LabView et le système à microcontrôleurs sont identiques, l'erreur entre les résultats de la simulation et ceux du système à microcontrôleurs a été nulle.

Tableau 4.1: Comparaison des indicateurs des signaux EMGdi simulés et expérimentaux (l = 100 ms, g = 2, fc = 30 Hz, ordre = 4)

	EMG brut	**Signal EMGdi simulé**	**Signal EMGdi expérimental**
ARV(ECG)	0.269	0.047	0.047
ARV(EMG)	0.40	0.34	0.34
SNR	1	4.86	4.86

L'égalité entre les paramètres ARV(ECG) et ARV(EMG) pour le signal simulé et expérimental montrent que l'atténuation des signaux ECG et la conservation des signaux EMG est resté la même après le traitement en simulation et le traitement expérimental.

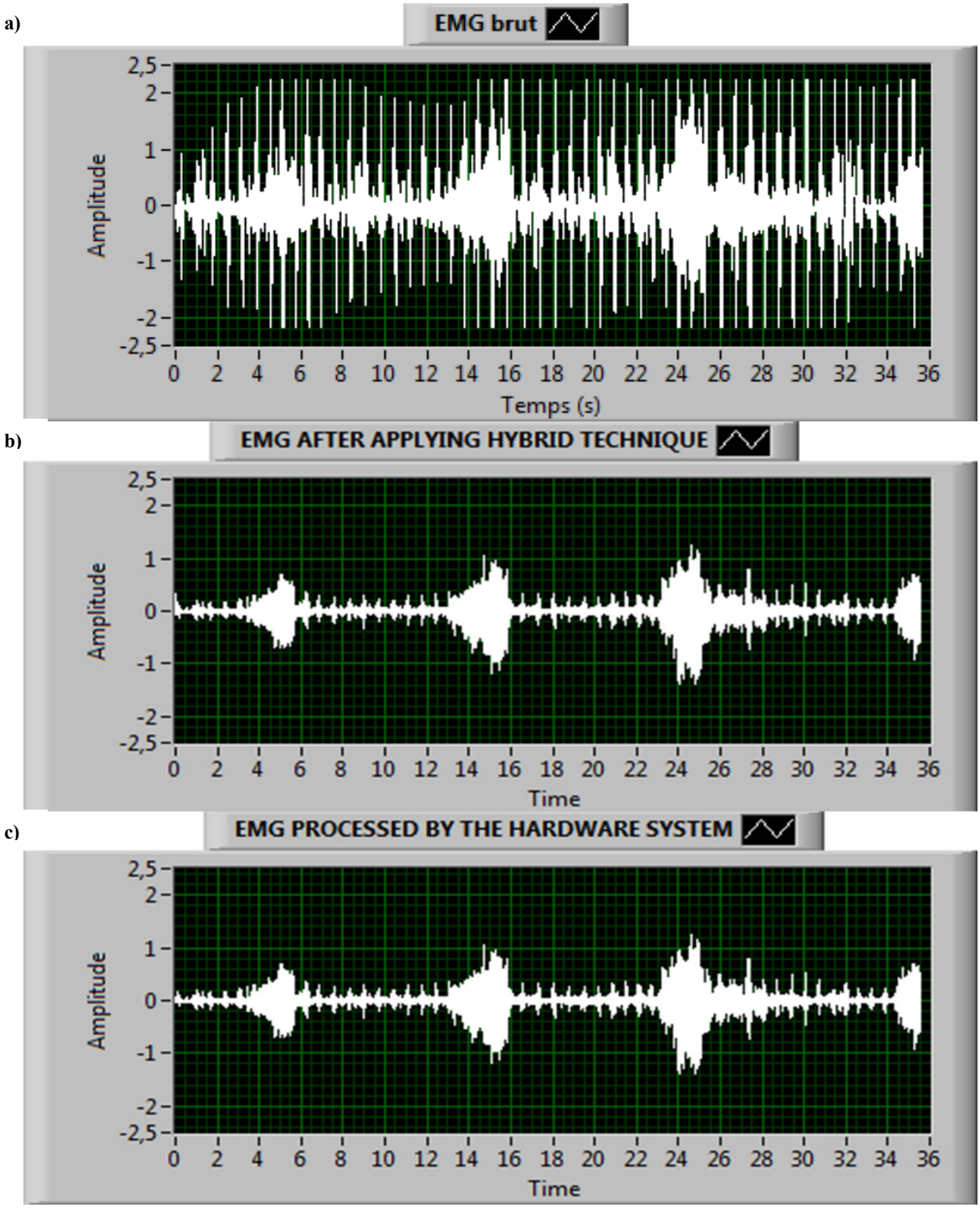


Figure 4.2 : a) EMGdi brut. b) EMGdi traité en simulation. c) EMGdi traité par le système à microcontrôleurs (l = 100 ms, g = 2, fc = 30 Hz, ordre = 4). Dans les graphes, le temps est exprimé en secondes et l'amplitude est une amplitude normalisée.

4.4 Vérification du fonctionnement de la nouvelle méthode NTH

Le code en langage LabView, réalisé dans le cadre de notre projet, implémente la technique de coupage, le filtrage passe-haut, la technique hybride MH ainsi que la nouvelle méthode NTH qu'on a proposée dans le chapitre 4 et qui permet d'éliminer complètement la contamination ECG pendant les périodes expiratoires tout en détectant l'activité musculaire. Étant donné que les résultats par simulation de la technique de coupage et du filtrage passe-haut ont été présentés dans le chapitre 2 dans le cadre de l'étude comparative des méthodes d'élimination des ECG en temps réel, les résultats obtenus pour ces deux méthodes ne seront pas présentés dans ce chapitre. Seuls les résultats obtenus par la méthode NTH proposée seront présentés dans cette section ainsi que les résultats obtenus pour la méthode MH pour des fins de comparaison.

La figure 4.3 a) montre les signaux EMGdi bruts utilisés dans le cadre de notre expérience. La figure 4.3 b) montre les résultats obtenus par la méthode MH qui seront comparés à ceux obtenus par l'approche NTH proposée. Comme on peut le constater dans la figure 4.4 a), la plupart de la contamination ECG pendant les périodes expiratoires, qui est juste atténuée par la technique MH (figure 4.3 b)), est éliminée lorsqu'on applique la méthode NTH proposée, et ceci tout en conservant l'information EMGdi. Ainsi, sur 32 contaminations ECG pendant les périodes expiratoires, seule une contamination n'a pas été détectée, ce qui correspond à un taux de réussite d'environ 97%.

La figure 4.4 b) montre les résultats de la détection automatique de l'activité du diaphragme en appliquant la méthode NTH. On peut constater que les quatre inspirations sont détectées. Cependant, on peut remarquer qu'au niveau de la troisième détection, il y a une légère

instabilité dans la détection vers la fin de l'inspiration. Ceci peut être expliqué par le fait que l'inspiration était restée faible pendant une durée relativement longue, ce qui n'a pas pu rendre évident le fait de distinguer la présence et l'absence de l'activité respiratoire.

Cependant, ces résultats n'ont été possibles qu'en utilisant une fenêtre de calcul de la moyenne ARV' égale à l' = 850 ms. En effet, plus la valeur de l' diminue, plus les performances de l'élimination des ECG pendant les périodes expiratoires diminue. D'un autre côté, plus on augmente la valeur de l', plus les pertes dans les signaux EMGdi augmente. Ainsi, la valeur optimale qu'on avait trouvée qui permet d'avoir les meilleurs résultats de détection de l'activité respiratoire et d'élimination de la contamination ECG pendant les périodes expiratoires sans pertes dans les signaux EMGdi était l' = 850 ms. La valeur du seuil de détection joue également un rôle important. La valeur optimale qu'on avait trouvée dans le cadre de nos tests était 0.068. Une valeur supérieure à cette valeur ne permettait pas d'éliminer toute la contamination ECG et une valeur inférieure à cette valeur impliquait des pertes dans les signaux EMGdi.

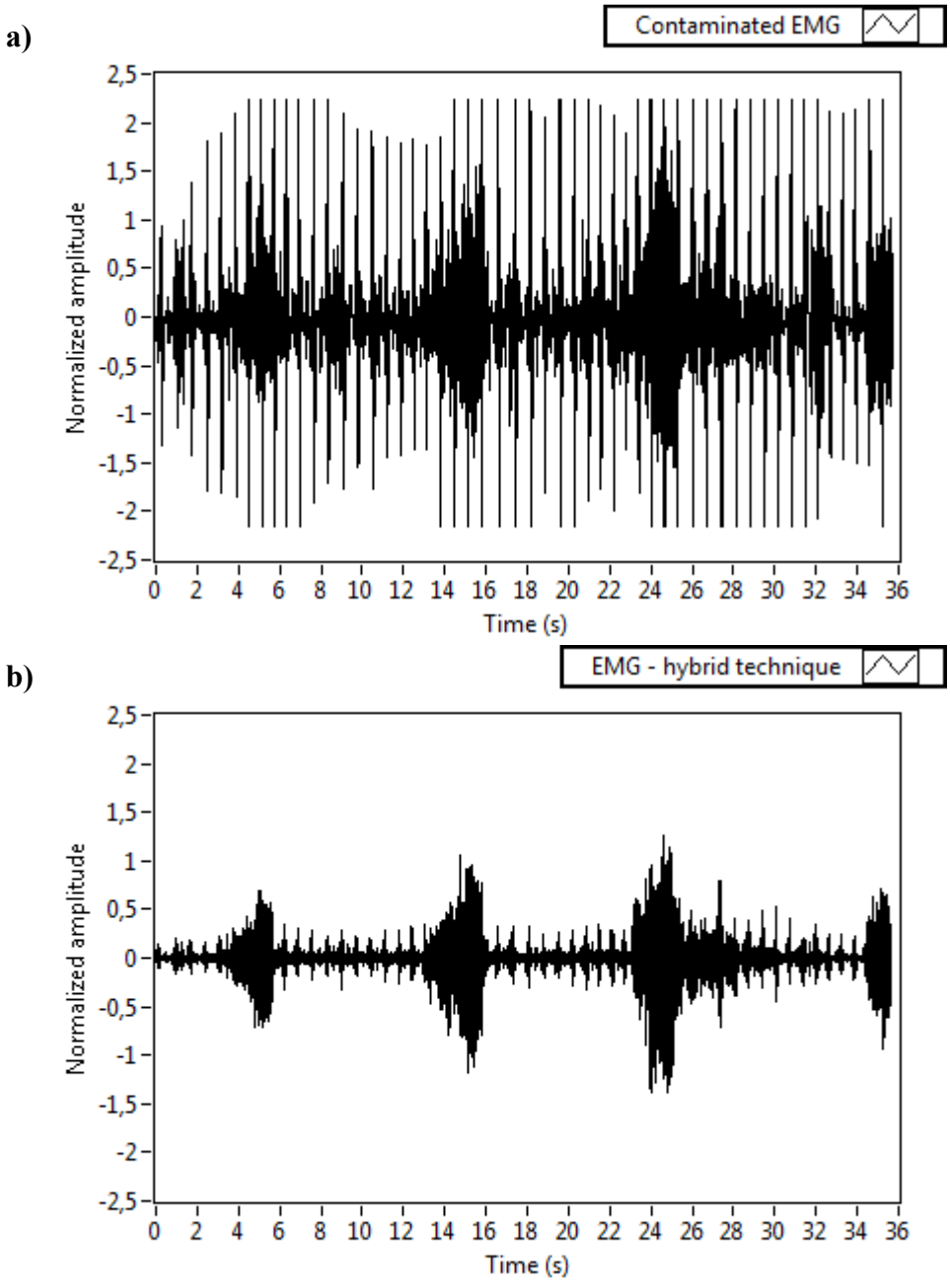


Figure 4.3: a) Signal EMGdi brut. b) Signal EMG après l'application de la méthode hybride MH

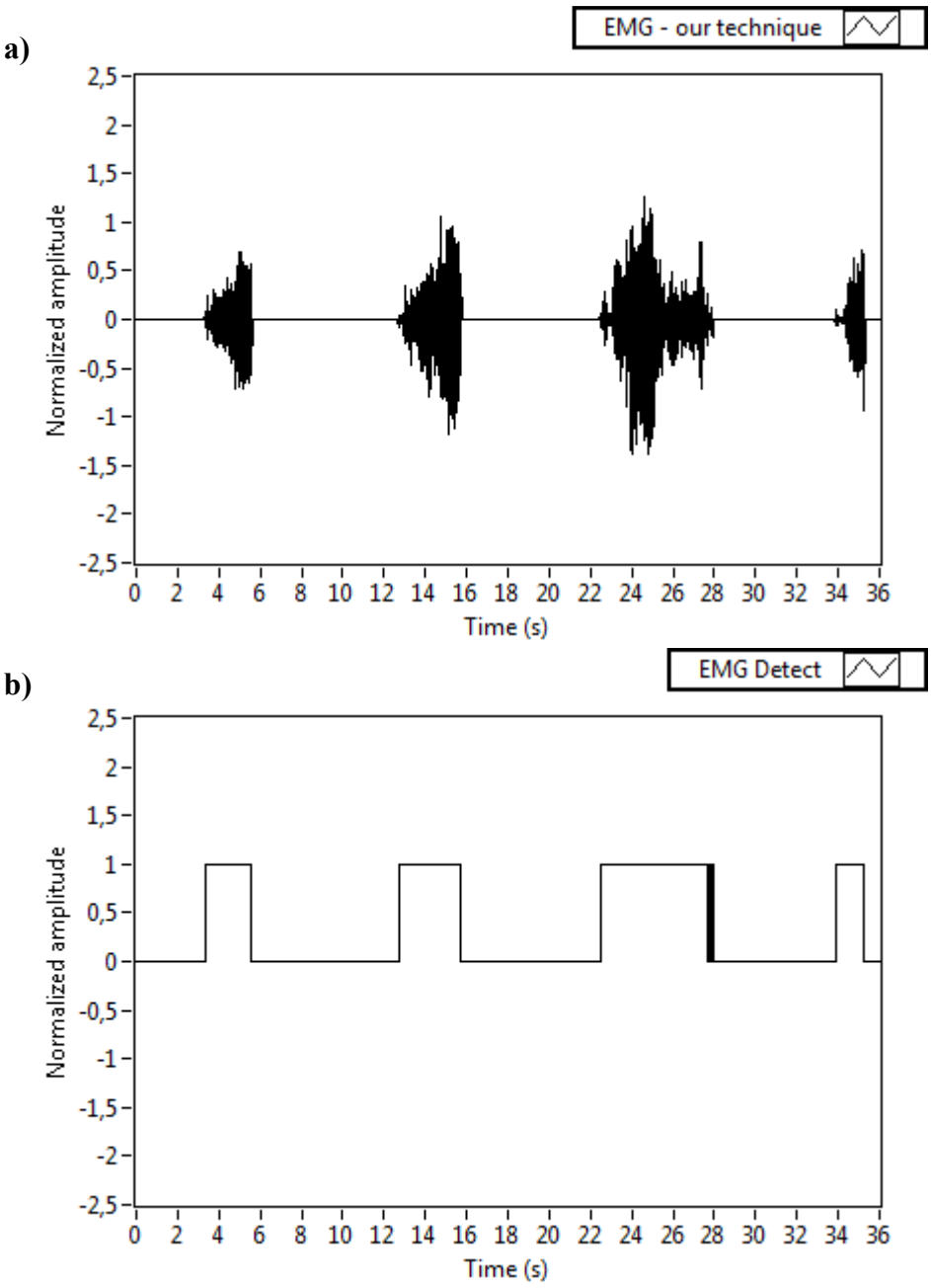


Figure 4.4 : a) signal EMGdi après application de notre méthode d'élimination des ECG pendant les périodes expiratoires NTH. b) Détection de l'activité musculaire du diaphragme. l' = 850 ms, seuil = 0.068, l = 100 ms, g =2, fc = 30 Hz, ordre = 4

4.5 Vérification du fonctionnement de l'architecture VHDL

4.5.1 Simulation présynthèse

Afin de pouvoir tester notre architecture VHDL implémentant l'algorithme Spike-Clipping (SC), un banc de test a été réalisé. Ce banc de test permet de récupérer les points EMGdi réels (figure 4.3 a) à partir d'un fichier texte et de les envoyer au programme VHDL qui réalise les traitements nécessaires et envoie le résultat dans un fichier de texte. Ce dernier fichier est récupéré par la suite et est affiché par le biais d'un programme LabView. Les résultats obtenus en simulation présynthèse sont présentés dans la figure 4.5 a. Comme on peut le constater le signal obtenu en simulation présynthèse par le programme VHDL est quasi-identique à celui obtenu en simulation LabView (figure 4.5 b). La différence de moins de 1% entre les deux courbes se justifie par la précision des calculs.

Les paramètres utilisés pour les simulations présynthèse et postsynthèse sont g = 2 et l = 200 ms.

La fréquence d'opération utilisée dans banc de test est de 20 MHz, ce qui permet de traiter un échantillon EMG toutes les 50 ns.

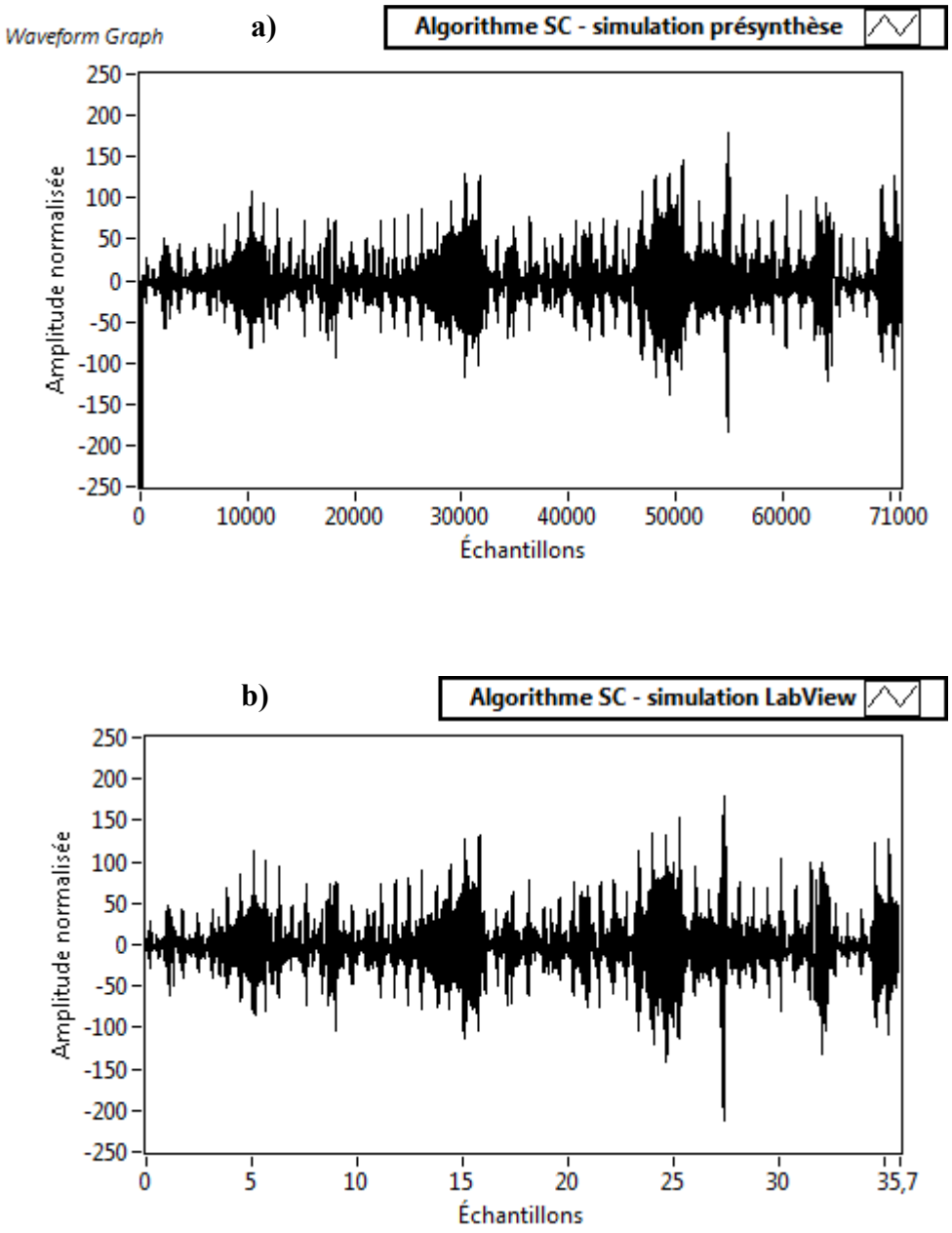


Figure 4.5 : a) Signal obtenu par la simulation présynthèse. b) Signal EMGdi obtenu par la simulation LabView

4.5.2 Simulation postsynthèse

La courbe EMGdi obtenue en simulation postsynthèse est présentée dans la figure 4.5. La synthèse a été réalisée par le biais du synthétiseur XST inclus dans le logiciel ISE 9.2 i de Xilinx et en utilisant la technologie Virtex II pro XC2VP70. Nous avons constaté que la courbe EMGdi de la simulation post synthèse est identique à la courbe obtenue en simulation Présynthèse dans la figure 4.4 a. Ceci montre le bon fonctionnement de notre architecture VHDL et permet de valider cette architecture.

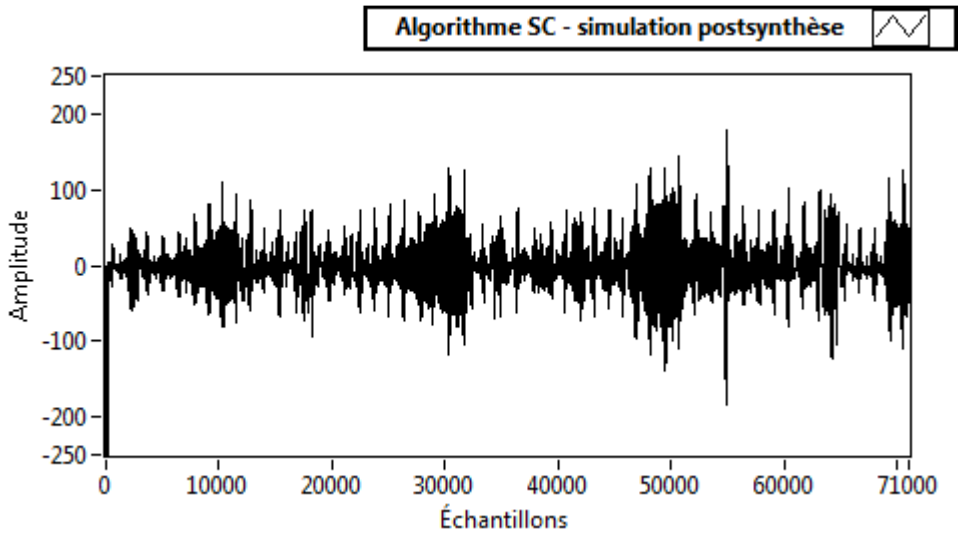


Figure 4.6 : Signal EMGdi obtenu par la simulation postsynthèse

4.6 Conclusion

Dans ce chapitre, nous avons montré et expliqué les résultats expérimentaux obtenus par l'implémentation de la méthode hybride (MH), adaptée pour convenir à des signaux EMGdi, sur notre système à microcontrôleurs. Les résultats obtenus ont démontré pour la première fois le

fonctionnement de cette technique sur un support matériel ainsi que le fonctionnement de cette technique sur des signaux EMG de type diaphragmatique.

Dans un second lieu, nous avons présenté dans ce chapitre les résultats obtenus en appliquant la nouvelle technique hybride, notée NTH, qu'on a développée dans le chapitre 3. Les résultats obtenus ont démontré que cette nouvelle méthode permettait d'avoir des résultats meilleurs que ceux obtenus par la technique hybride classique tout en permettant de détecter, en temps réel, l'activité musculaire du diaphragme.

Finalement, nous avons présenté les résultats présynthèse et postsynthèse obtenus pour l'architecture VHDL implémentant l'algorithme SC. Ces résultats, identiques aux résultats obtenus en simulation LabView, valident le bon fonctionnement de cette architecture.

CHAPITRE 5 CONCLUSION ET RECOMMANDATIONS

Le présent mémoire de Maîtrise peut être divisé en trois parties. Tout d'abord, la première partie aborde une étude comparative détaillée des méthodes d'élimination, en temps réel, de la contamination ECG dans les signaux EMG ainsi qu'une brève comparaison des méthodes de détection de l'activité musculaire en temps réel. La seconde partie concerne l'implémentation matérielle et en langage VHDL de la méthode hybride, notée MH. Finalement, la troisième partie concerne la proposition d'une nouvelle technique hybride, notée NTH, basée sur l'approche MH, permettant, en plus du filtrage fourni par la technique MH, d'éliminer complètement la contamination ECG pendant les périodes de relâchement musculaire tout en détectant l'activation musculaire en temps réel. Ces trois parties du projet sont complémentaires. En effet, l'étude comparative nous a permis de trouver la méthode la plus efficace d'élimination des ECG dans les EMG en temps réel et nous a permis de connaitre les caractéristiques et les limites des techniques de détection de l'activité musculaire. Par la suite, étant donné que la technique hybride n'a jamais été testée sur des signaux EMG de type diaphragmatiques acquis via un cathéter œsophagien, ni testée sur une interface matérielle, la validation logicielle et matérielle de cette méthode sur des signaux EMGdi a été nécessaire. D'un autre côté, l'architecture VHDL proposée a constitué une première implémentation dédiée à la technique MH. Finalement, la troisième partie a constitué une contribution additionnelle aux parties précédentes et a tiré parti des différents éléments abordés dans les deux premières parties.

Les sections suivantes présentent les conclusions pour chaque partie abordée.

5.1 Étude comparative

Afin de trouver une méthode d'élimination de la contamination ECG dans les signaux EMGdi pouvant être implantée dans le système d'acquisition des signaux respiratoires présenté dans le premier chapitre, nous avons choisi parmi les méthodes existantes celles pouvant convenir à notre application. Notre choix s'est finalement porté sur trois méthodes : la technique de coupage, le filtrage passe-haut et la méthode hybride (MH). Afin de comparer ces trois méthodes, nous avons du adapter leurs paramètres afin de convenir aux signaux EMGdi acquis via notre système d'acquisition. Par la suite, nous avons implémenté ces méthodes en langage LabView et nous pu constater, d'après les résultats en simulation obtenus et les paramètres de comparaison choisis, que la méthode MH, était la plus performante et la mieux adaptée pour notre application.

D'autre part, étant donné que l'une des applications futures de notre système d'acquisition est le contrôle automatique de la respiration artificielle en se basant sur les signaux EMGdi acquis, nous avons présenté une comparaison des différentes méthodes de la détection automatique de l'activité musculaire basée sur les signaux EMG. Cette comparaison nous a permis de mettre en évidence les avantages et les inconvénients de ces méthodes afin de proposer plus tard une technique plus avantageuse.

5.2 Implémentation de la méthode hybride MH

Étant donné que la technique MH a été uniquement testée en simulation dans les travaux précédents, et étant donné que cette technique a été uniquement testée sur des signaux EMG thoraciques surfaciques, nous avons implémenté cette technique sur un système à microcontrôleurs après avoir modifié la méthode de calcul de la moyenne ARV (Average Rectified Value) pour accélérer les calculs, et après avoir ajusté les paramètres de cette technique, notamment la fréquence de coupure, le gain et la longueur de fenêtrage, pour convenir au type des signaux EMG que nous utilisons. Ce système a été choisi de telle sorte qu'une intégration des programmes implémentés dans les microcontrôleurs du système d'acquisition des signaux respiratoires soit plus facile. Les résultats expérimentaux ont démontré que les deux implémentations logicielle et matérielle donnent les mêmes performances.

Par la suite, nous avons proposé une nouvelle architecture VHDL fonctionnelle dédiée à la technique MH. Ceci rendra possible dans le futur l'intégration d'un circuit numérique implémentant la technique MH dans le système d'acquisition des signaux respiratoires de notre équipe Polystim.

5.3 Nouvelle méthode de filtrage et de détection

Nous avons tenté de dépasser les performances obtenues par la technique MH et de développer une méthode élaborée permettant à la fois d'offrir, en temps réel, le même résultat que la méthode MH pendant les périodes inspiratoires et d'éliminer complètement la

contamination ECG pendant les périodes expiratoires tout en détectant l'activation du diaphragme. Ainsi, nous avons rajouté à la technique hybride adaptée un bloc de traitement et de détection permettant d'éliminer le reste de la contamination ECG pendant les périodes expiratoires et en exploitant cette reconnaissance des périodes expiratoires pour détecter l'activation musculaire du diaphragme. Les résultats obtenus ont démontré clairement que la méthode qu'on propose, notée NTH, offrait des résultats meilleurs que ceux obtenus par la technique hybride tout en permettant d'offrir une reconnaissance de l'inspiration supérieure à celle des méthodes existantes de détection de l'inspiration.

5.4 Recommandations

La méthode hybride (MH) a été validée sur des signaux EMGdi provenant d'un sujet en bonne santé respiratoire. Afin d'explorer les limites de cette technique, il serait nécessaire de la tester sur des signaux EMGdi provenant d'un sujet souffrant de fatigue diaphragmatique, par exemple.

De plus, puisque la technique MH a été validée matériellement, il serait intéressant de l'intégrer directement dans la carte de contrôle du système d'acquisition des signaux respiratoires de l'équipe Polystim. Cependant, il serait encore plus intéressant de valider matériellement le fonctionnement de la nouvelle méthode NTH que nous avons proposé et de l'intégrer par la suite au système d'acquisition.

Nous avons pu remarquer dans le chapitre 4 que, lorsque l'inspiration est faible et constante pendant une durée de temps relativement longue, la détection de l'inspiration devient instable et la partie du signal EMGdi pendant cette durée pourrait être considérée comme une

expiration. Bien que cela soit un inconvénient sans une importance notable dans le cadre de notre application, il serait intéressant d'y remédier afin d'exploiter cette méthode dans des applications différentes et pour des signaux EMG de types différents. Il serait également intéressant de superposer le signal EMGdi avec le signal Pdi pour avoir un indicateur précis du commencement et de la fin de la respiration.

Finalement, la nouvelle méthode NTH que nous avons proposée pourrait théoriquement être applicable à des EMG servant à diverses applications comme le contrôle des prothèses myoélectriques ou le contrôle à distance d'instruments en se basant sur le mouvement des mains. Le logiciel LabView et le système à microcontrôleurs développés pourront être utilisés pour valider cette méthode pour diverses applications puisqu'ils ne dépendent pas de la nature des signaux utilisés.

RÉFÉRENCES

Abbink, J. H., Van Der Bilt, A., et Van Der Glas, H. W. (1998). Detection of onset and termination of muscle activity in surface electromyograms. *J. Oral. Rhabil., 25*, 365-369.

Akkiraju, P., et Reddy, D. C. (1992). Adaptive cancellation technique in processing myoelectric activity of respiratory muscles. *IEEE Transactions On Biomedical Engineering* (Vol. 39, pp. 652-655).

Bartolo, A., Dzwonczyk, R. R., Roberts, C., et Goldman, E. (1996). Description and validation of a technique for the removal of ECG contamination from diaphragmatic EMG signal. *Medical & Biological Engineering & Computing, 34*, 76-81.

Basmajian, J. V., et De Luca, C. J. (1985). *Muscles Alive: Their Functions Revealed by Electromyography*: Williams & Wilkins.

Bloch, R. (1983). Subtraction of electrocardiographic signal from respiratory electromyogram. *Journal of Applied Physiology, 55*, 619-623.

Bonato, P., D'Alessio, T., et Knaflitz, M. (1998). A statistical method for the measurement of muscle activation intervals from surface myoelectric signal during gait. *IEEE Transactions on Biomedical Engineering, 45*, 287-298.

Cao, Y., Chen, C., et Hu, Y. (2005). Application of independent component analysis to ECG cancellation in surface electromyography measurement. *J. Biomed. Eng., 22*, 686-689.

Chabot, E., DiCecco, J., et Sun, Y. (2006). Microprocessor based control of electromechanical devices by using electromyogram: a "cricket car" model. *IEEE Northeast Bioengineering conference* (Vol. 13, pp. 135-163).

Chen, J. D. Z., Lin, Z. Y., Ramahi, M., et Mittal, R. K. (1994). Adaptive cancellation of ECG artifacts in the diaphragm electromyographics signals obtained through intraoesophageal electrodes during swallowing and inspiration *Neurogastroenterol. Mot., 6*, 279-288.

De Luca, C. J. (1979). Physiology and mathematics of myoelectric signals. *IEEE Transactions on Biomedical Engineering, 26*, 313-325.

Désilets, T., Sawan, M., et Bellemare, F. (2006). Wireless esophageal catheter dedicated to respiratory diseases diagnostic. *IEEE-ISCAS* (pp. 2581-2584).

Dido, J. (2002). *Système d'acquisition de la pression transdiaphragmatique et de l'EMGdi.* École Polytechnique de Montréal, Montreal.

Donoho, D. L., et Johnstone, I. M. (1994). Ideal spatial adaptation via wavelet shrinkage. *Biometrika, 81*, 425-455.

Dow, D. E., Mantilla, C. B., Zhan, W.-Z., et Sieck, G. C. (2006). EMG-based detection of inspiration in the rat diaphragm muscle. *IEEE-EMBS* (pp. 1204-1207).

Hodges, P. W., et Bui, B. H. (1996). A comparison of computer-based methods for determination of onset of muscle contraction using electromyography. *Electroenceph. Clin. Neurophysiol., 101*, 511-519.

Hu, Y., Mak, J., Hongtao, L., et DK Luk, K. (2007). ECG cancellation for surface electromyography measurement using independent component analysis. *IEEE-ISCAS* (pp. 3235-3238).

Levine, S., Gillen, J., Weiser, P., Gillen, M., et Kwatny, E. (1986). Description and validation of an ECG removal procedure for EMGdi power spectrum analysis. *Journal of Applied Physiology, 60*, 1073-1081.

Lidierth, M. (1986). A computer based method for automated measurement of the periods of muscular activity from an EMG and its application to locomotor EMGs. *Electroenceph. Clin. Neurophysiol., 64*, 378-380.

Loring, S. H., et Malhotra, A. (2007). Inspiratory efforts during mechanical ventilation: is there a risk of barotrauma? *Chest, 131*, 646-648.

Marque, C*., et al.* (2005). Adaptive filtering for ECG rejection from surface EMG recordings. *J. of Electromyogr. Kinesiol., 15*, 310-315.

Mosby. (2009). Mosby's Medical Dictionary. In Elsevier (Ed.), *Mosby's Medical Dictionary* (8th ed.).

Mrvaljevic, N., Zaczynski, R., Chabot, E., et Sun, Y. (2007). Microprocessor based algorithms for controlling electromyogram-driven devices. *IEEE-NEBC*.

Redfern, M. S., Hughes, R. E., et Chaffin, D. B. (1993). High-pass filtering to remove electrocardiographic interference from torso EMG recordings. *Clin. Biomech., 8*, 44-48.

Rhou, B., Sawan, M., Désilets, T., et Bellemare, F. (2008). Real-time filtering technique to remove ECG interference from recorded esophageal EMG. *IEEE-Biocas*.

Schweitzer, T. W., Fitzgerald, J. W., Bowden, J. A., et Lynne-Davies, P. (1979). Spectral analysis of human inspiratory diaphragmatic electromyograms. *Journal of Applied Physiology, 46*, 152-165.

Sinderby, C*., et al.* (2007). Inspiratory muscle unloading by neurally adjusted ventilatory assist during maximal inspiratory efforts in healthy subjects. *Chest, 131*, 711-717.

Straude, G., Flachenecker, C., Daumer, M., et Wolf, W. (2001). Onset detection in surface electromyographic signals: a systematic comparison of methods. *EURASIP Journal on Applied Signal Processing, 2*, 67-81.

Straude, G., et Wolf, W. (1999). Objective motor response onset detection in surface myoelectric signals. *Med. Eng. Phys., 21*, 449-467.

Tavernier, C. (2002). *Microcontrôleurs AVR : description et mise en œuvre*. Paris: DUNOD.

Webster, J. G. (1998). *Medical instrumentation : application and design* (3^{e} éd.): John Wiley & Sons.

Widrow, B*., et al.* (1975). Adaptive noise cancelling: principles and applications. *Proc. IEEE, 63*, 1692-1716.

Zecca, M., Micera, S., Carrozza, M. C., et Dario, P. (2002). Control of multifunctional prosthetic hands by processing the electromyographic signal. *Critical Reviews in Biomedical Engineering, 30*, 459-485.

Zhou, P., et Kuiken, T. A. (2006). Eliminating cardiac contamination from myoelectric control signals developed by targeted muscle reinnervation. *Physiological Measurement, 27*, 1311-1327.

Zhou, P., Lock, B., et Kuiken, T. A. (2007). Real time ECG artifact removal for myoelectric prosthesis control. *Physiological Measurement, 28*, 397-413.

ANNEXE A – Origine des signaux EMG

Le signal EMG est un enregistrement graphique de l'activité électrique intrinsèque d'un muscle du corps. Cette activité électrique est le résultat de l'activité électrique générée par une unité appelée unité motrice (figure A.1). L'unité motrice est composée d'un neurone moteur et d'un groupe de fibres musculaires innervées par ce neurone. Ces fibres musculaires, de diamètre se situant entre 10 et 100 micromètres et de longueur allant de quelques millimètres à 30 centimètre [Basmajian et De Luca, 1985], sont regroupées en plusieurs groupes qui composent le muscle et sont responsables des mouvements de contraction et de décontraction musculaire.

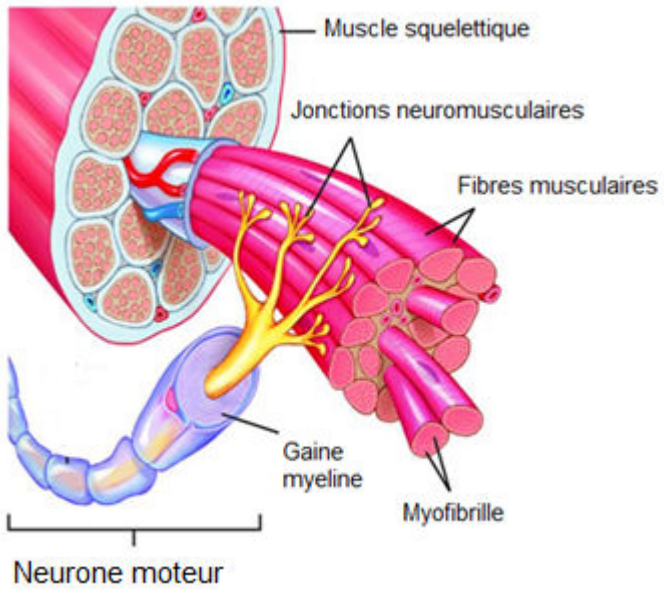


Figure A.1 : Représentation d'une unité motrice [Mosby, 2009].

Lors d'un effort volitif relié à un muscle donné, les neurones moteurs reliés à ce muscle transmettent un signal électrique à un groupe de fibres musculaires, impliquant une activation synchrone de toutes ces fibres. Cependant, les différents groupes de fibres constituant un

muscle donné ne sont pas synchrones. En effet, juste avant la contraction musculaire, les unités motrices de plus petite taille sont activées en premier, et au fur et à mesure de la contraction, les unités motrices de plus grandes tailles sont activées d'une manière croissante. Ceci explique le fait que l'amplitude d'un signal EMG s'accroit au fur et à mesure de l'accroissement de l'effort de contraction.

Lorsqu'une unité motrice est activée, un faible courant électrique parcourt les fibres de cette unité motrice générant un potentiel d'action appelé MUAP [De Luca, 1979].

Le signal EMG est constitué de l'ensemble des signaux électriques générés par tous les MUAP dans la zone de détection de l'électrode utilisé pour l'acquisition d'un EMG.

ANNEXE B – Diviseur binaire

Le diviseur implémenté dans notre architecture VHDL, présentée dans le chapitre 3, est représenté dans la figure B.2. Ce diviseur permet d'effectuer la division de deux nombres binaires : un dividende de 25 bits, représentant la somme des N échantillons EMG précédents rentrant dans le calcul de la moyenne ARV lors de l'application de l'algorithme SC, et un diviseur représentant le nombre d'échantillons inclus dans la fenêtre de calcul de la moyenne ARV. Ce diviseur permet d'avoir un résultat de 12 bits représentant la moyenne ARV. Les cellules C (figure B.2) sont détaillées dans la figure B.1.

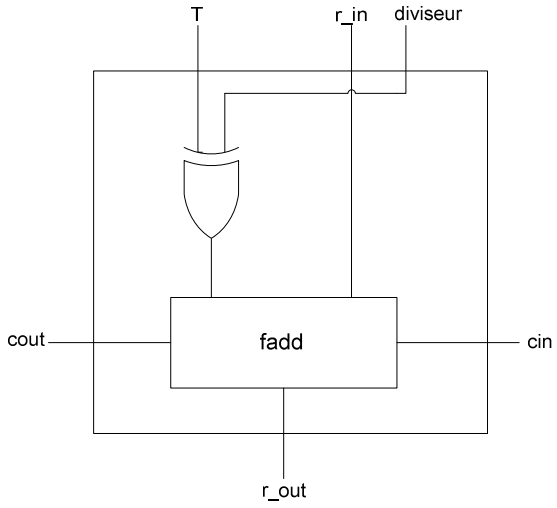


Figure B.1 : Schéma détaillé de la cellule C.

Le bloc fadd permet de calculer r_out et cout à partir des relations suivantes :

r_out = T xor r_in xor diviseur xor cin

cout = (T xor diviseur and r_in) or (T xor diviseur and cin) or (r_in and cin)

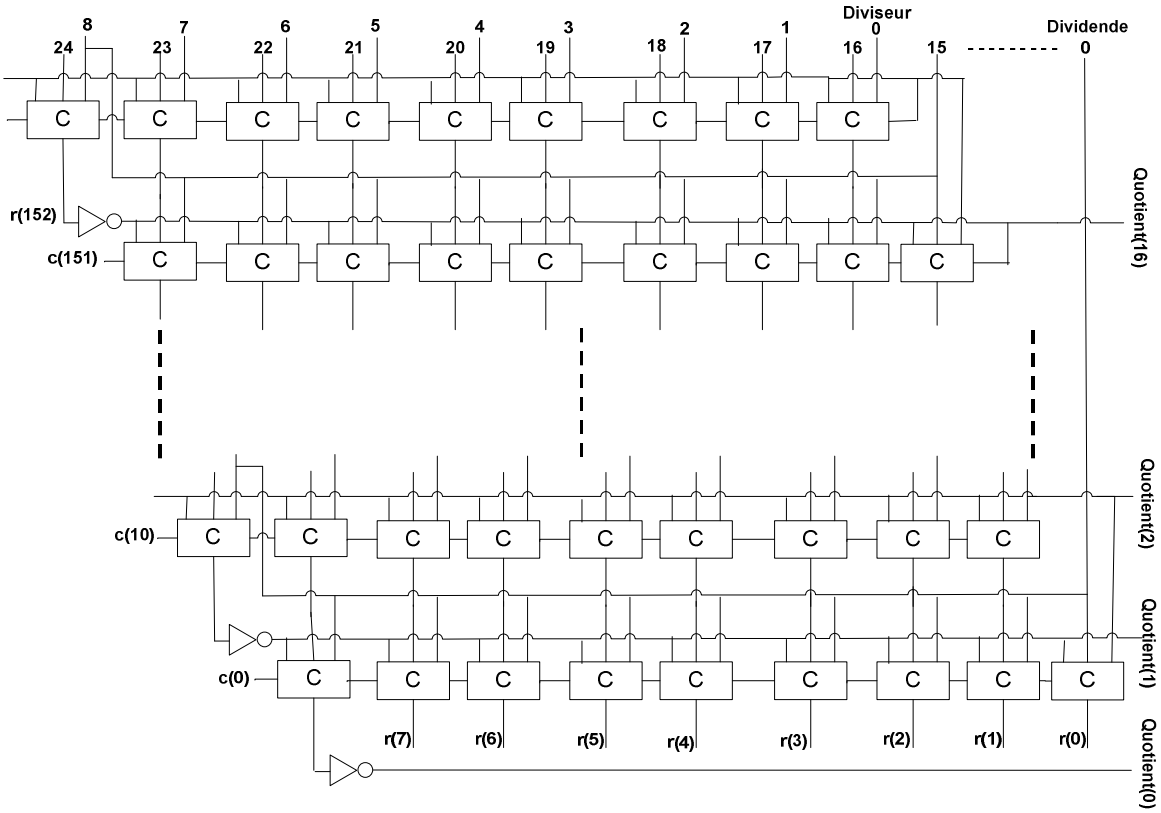
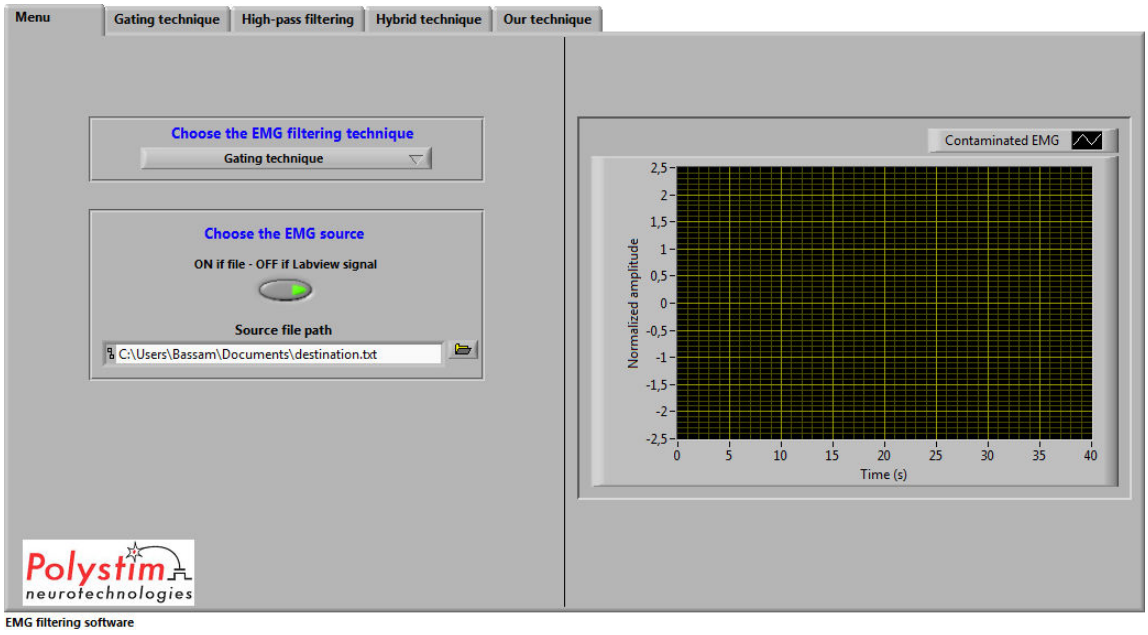


Figure B.2 : Schéma du diviseur binaire.

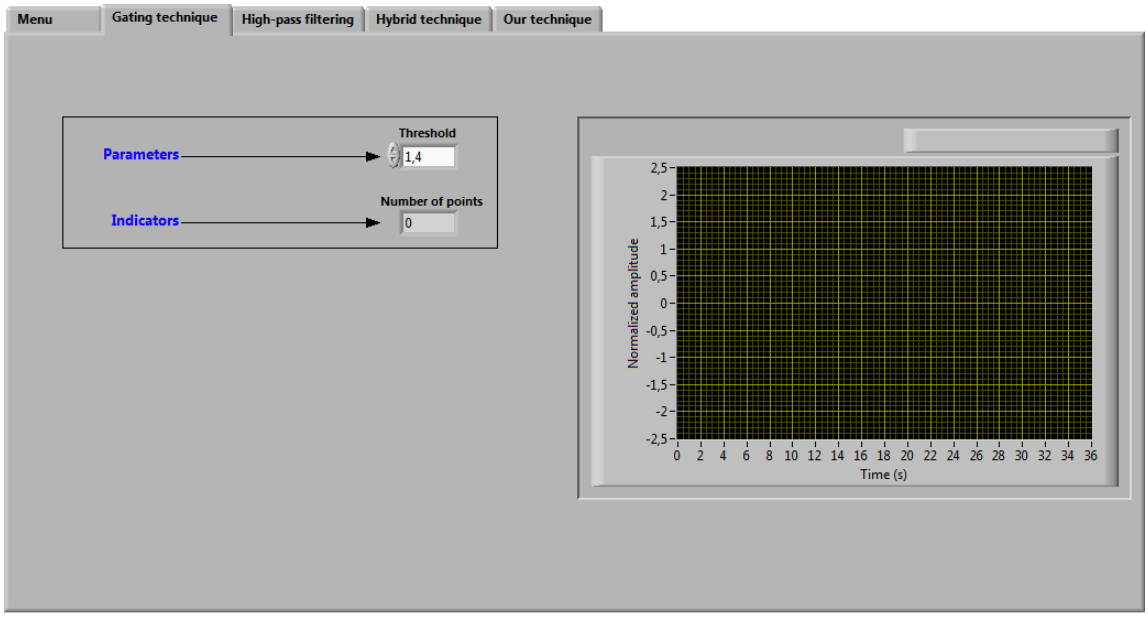
ANNEXE C – Logiciel réalisé en langage LabView

I- IHM du logiciel de traitement des signaux EMG par diverses méthodes

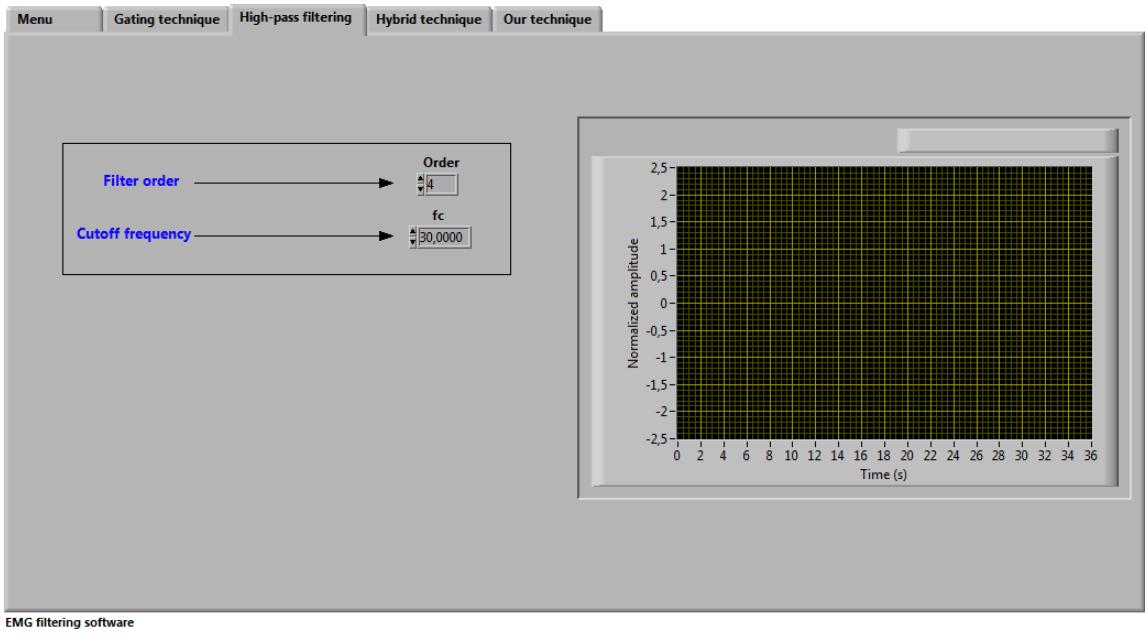
1. Onglet menu



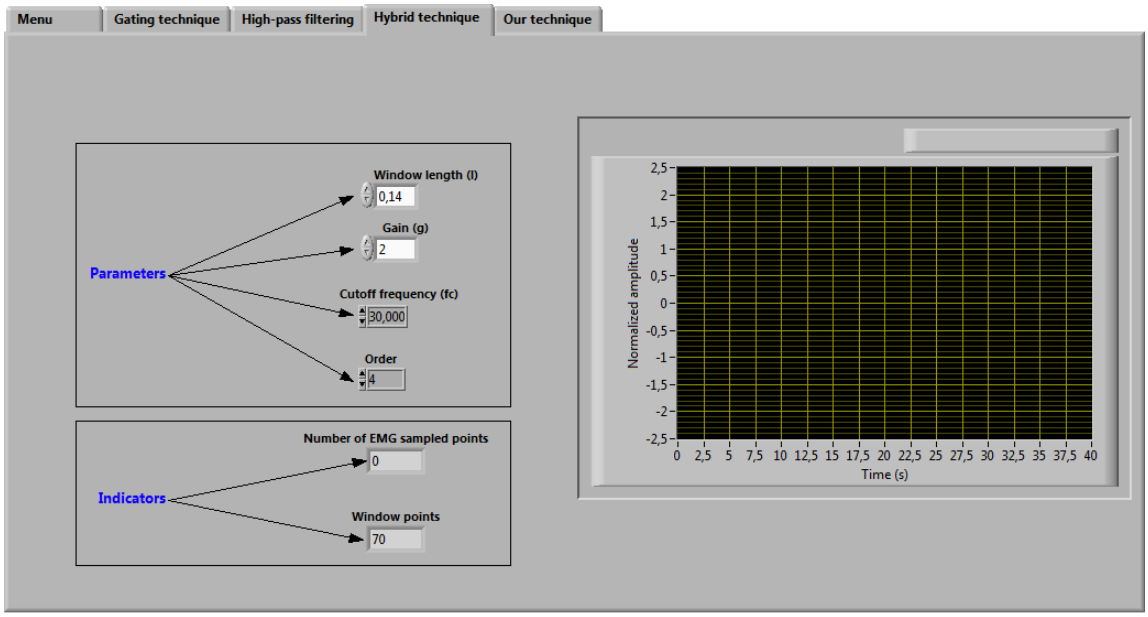
2. Onglet technique de coupage



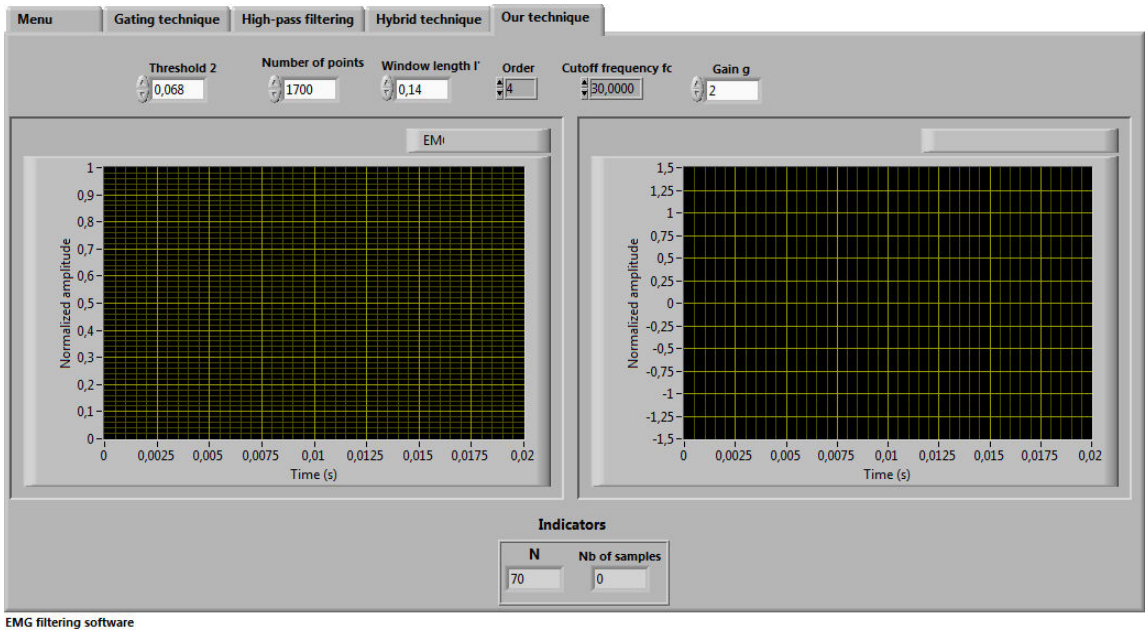
3. Onglet filtrage passe-haut



4. Onglet technique hybride



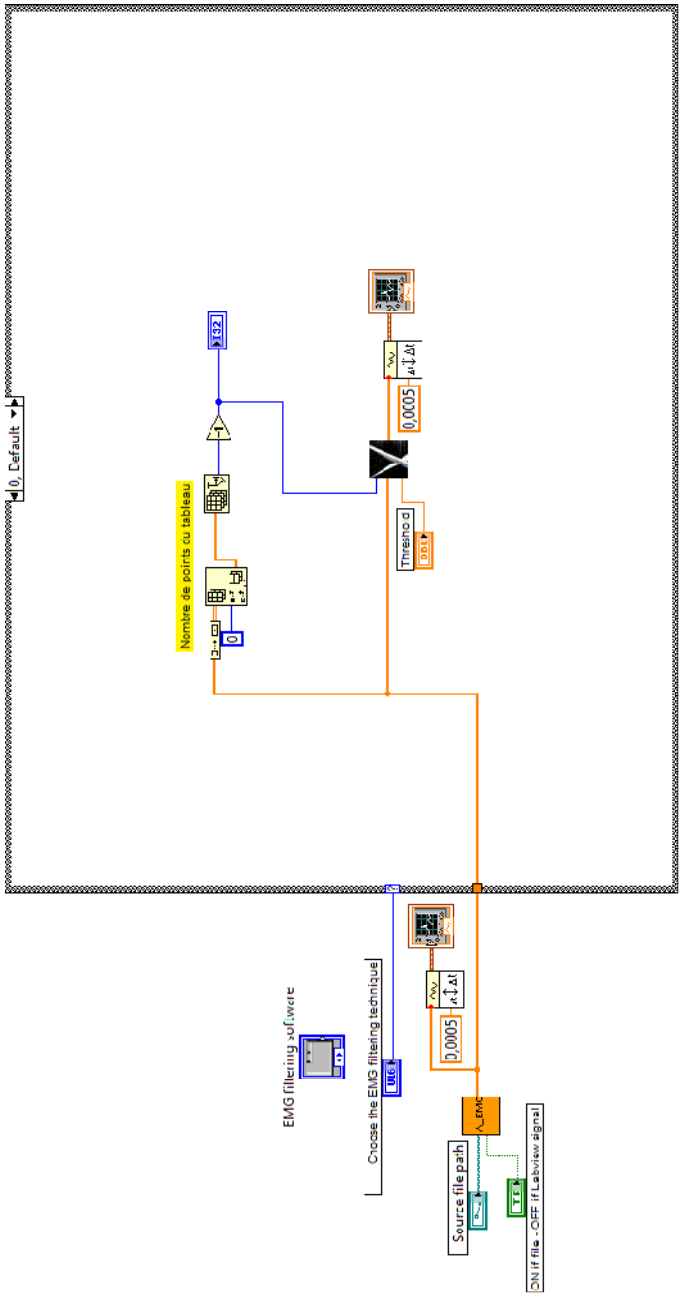
5. Onglet méthode hybride proposée



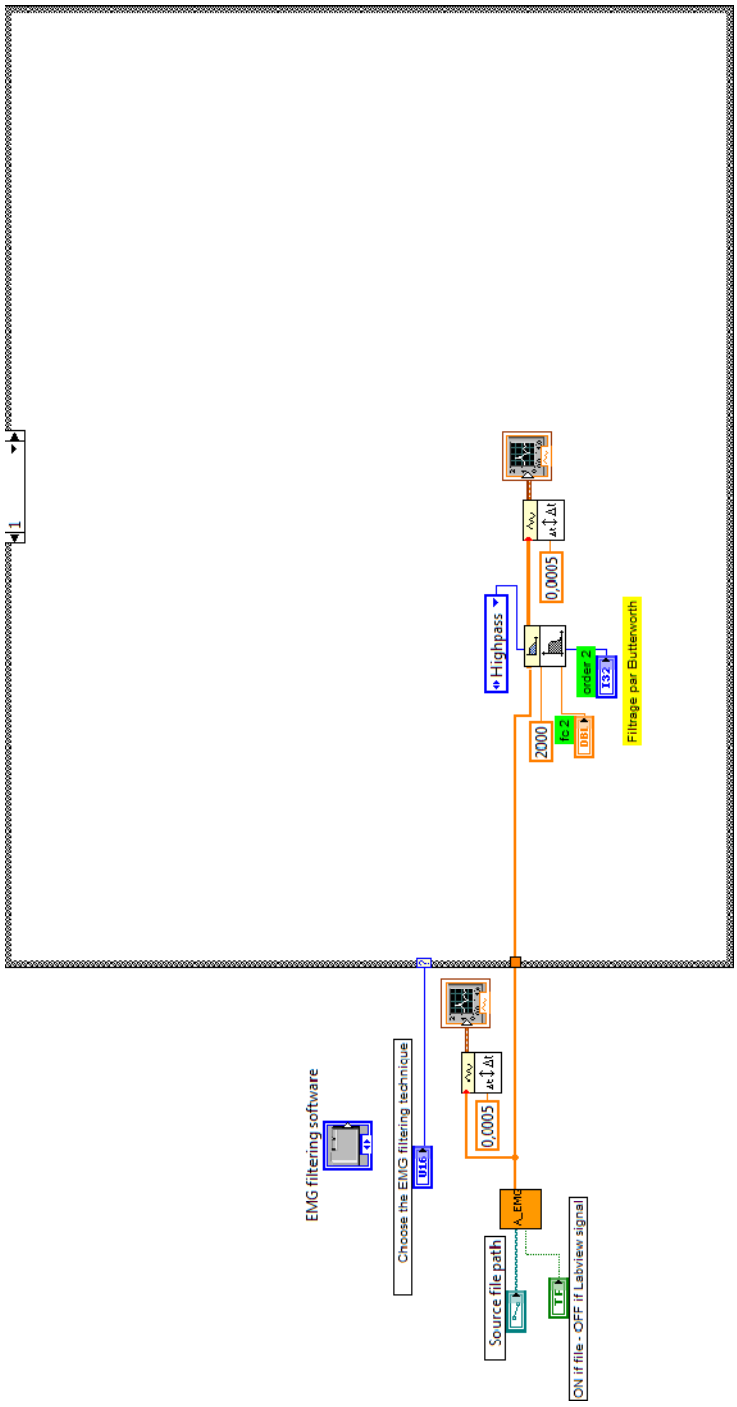
II- Code LabView du logiciel de traitement des signaux EMG

1. VIs du programme principal

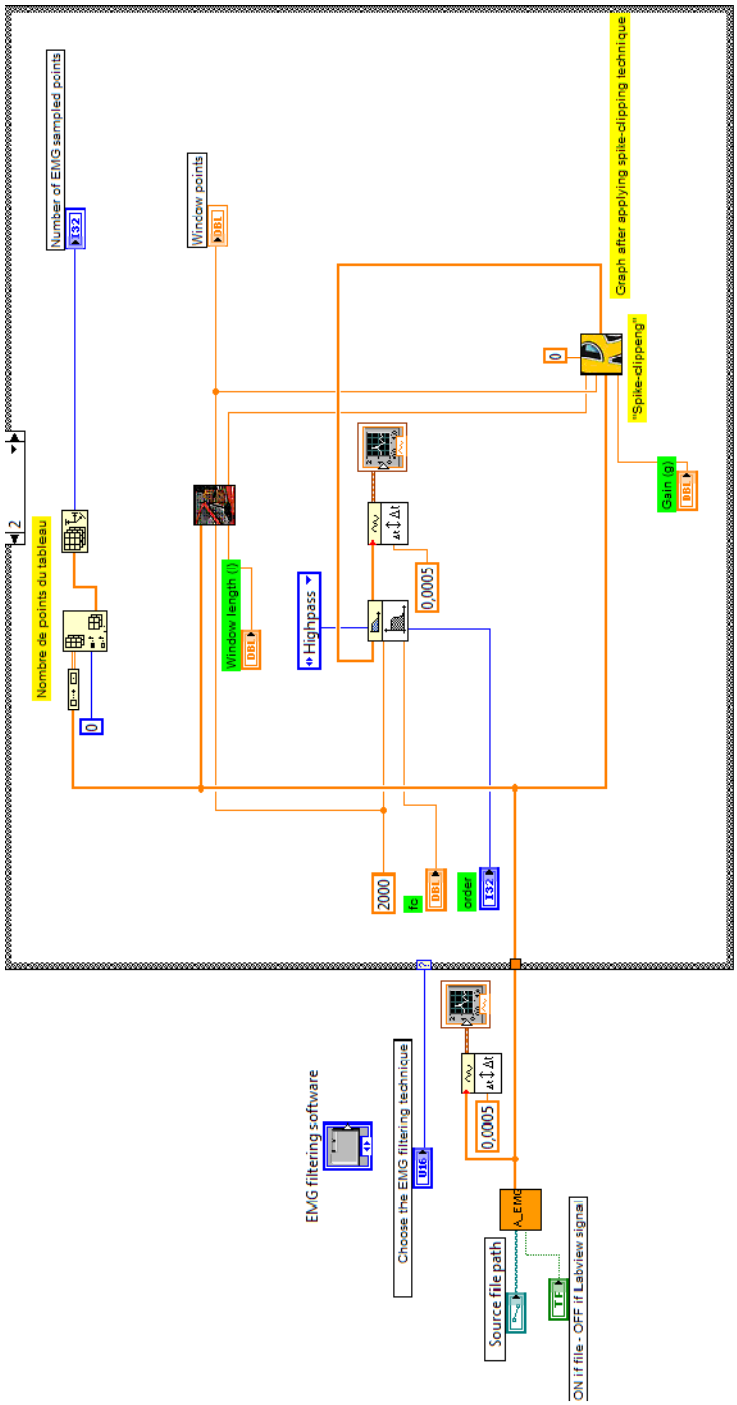
1.1 VI de l'onglet de la technique de coupage



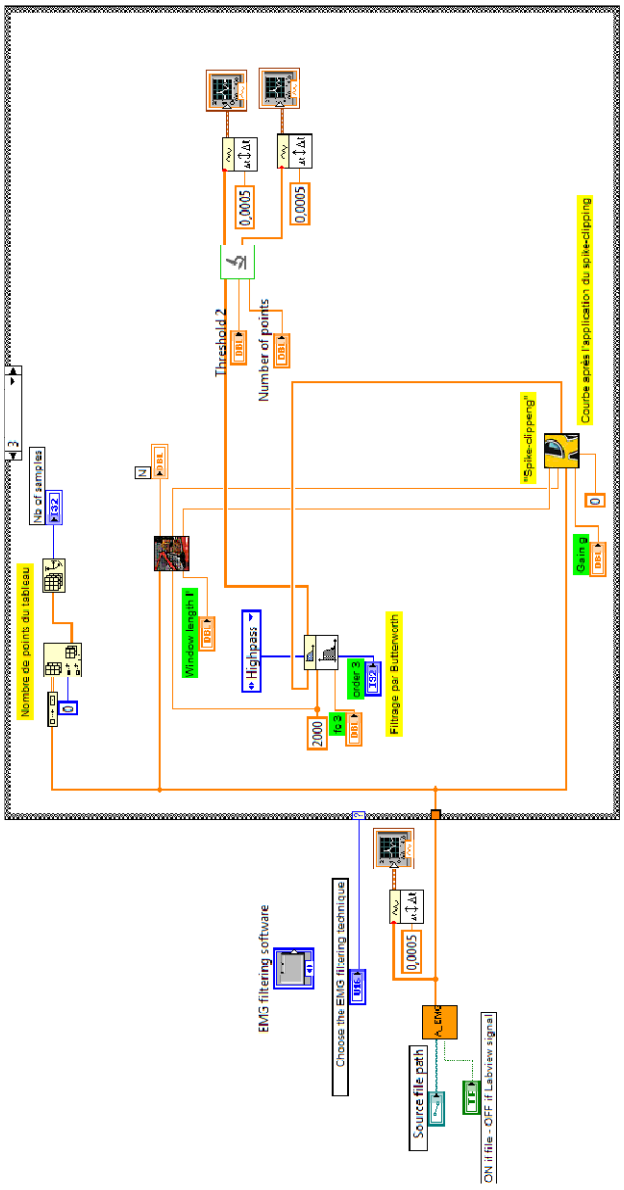
1.2 VI de l'onglet du filtrage passe-haut



1.3 VI de l'onglet de la technique hybride

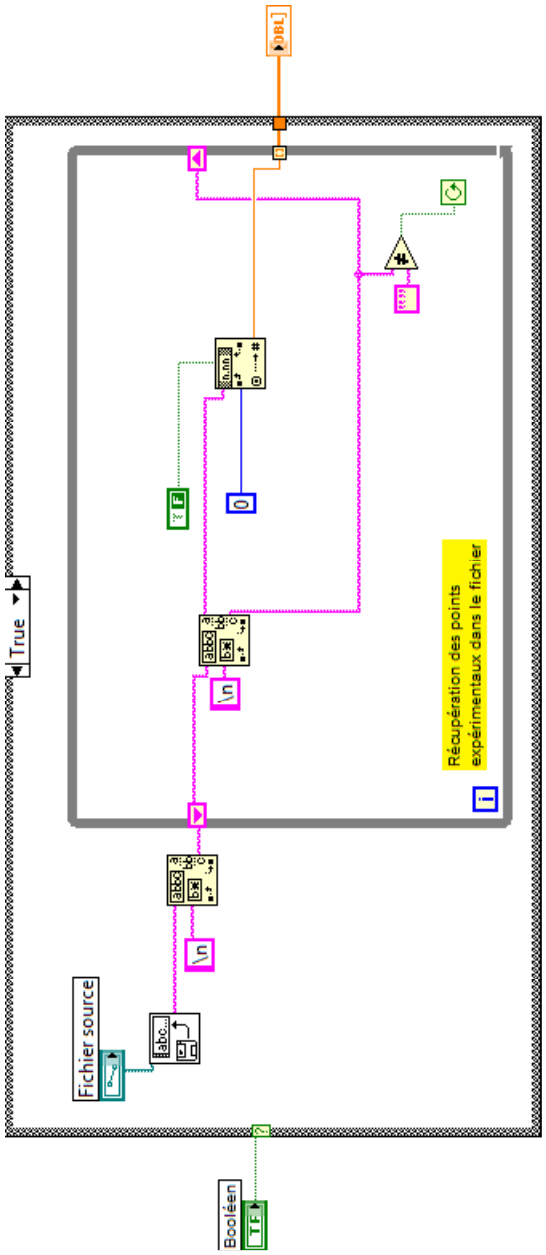


1.4 VI de l'onglet de la méthode hybride proposée

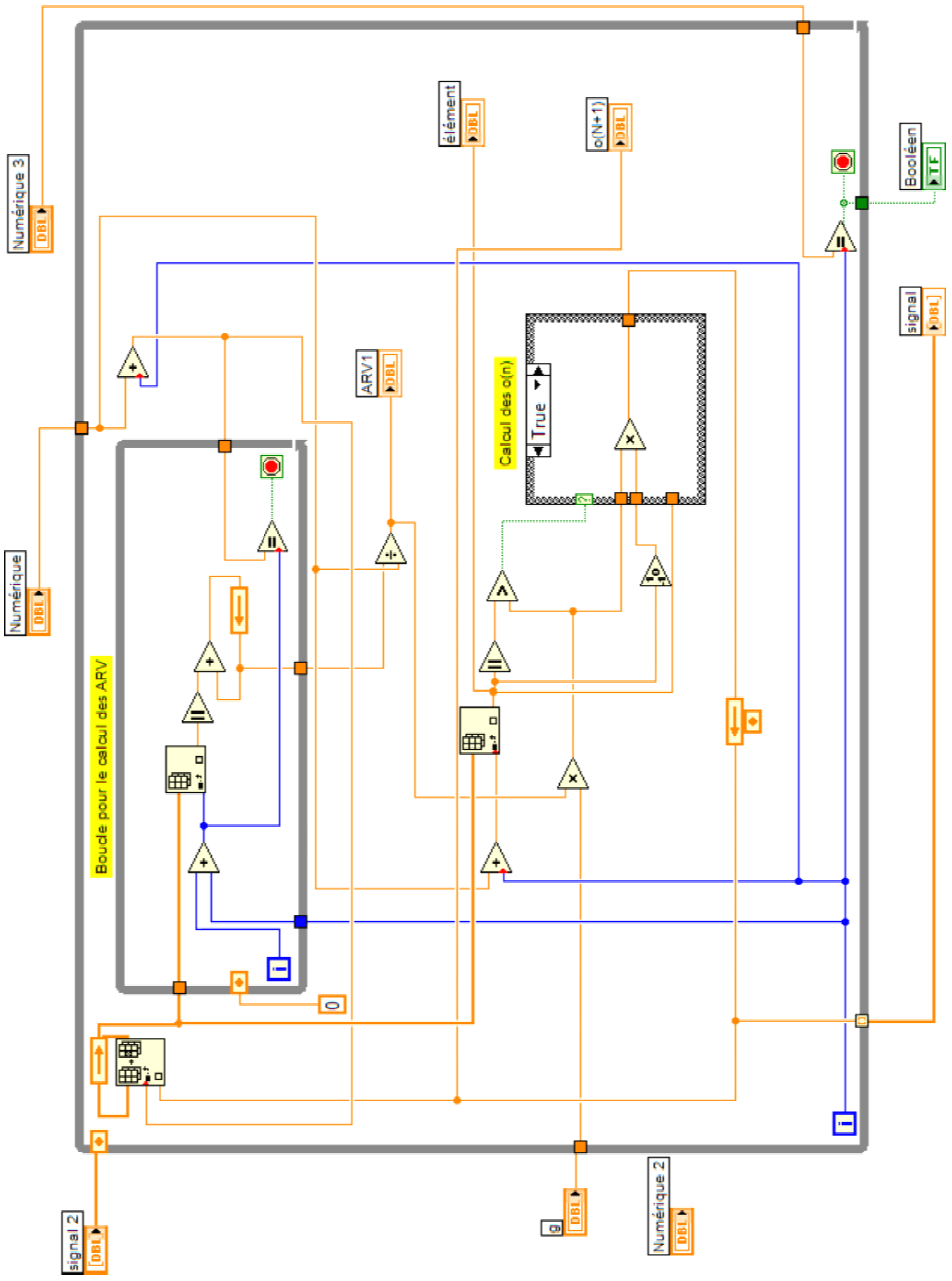


2. VIs des différentes fonctionnalités du logiciel

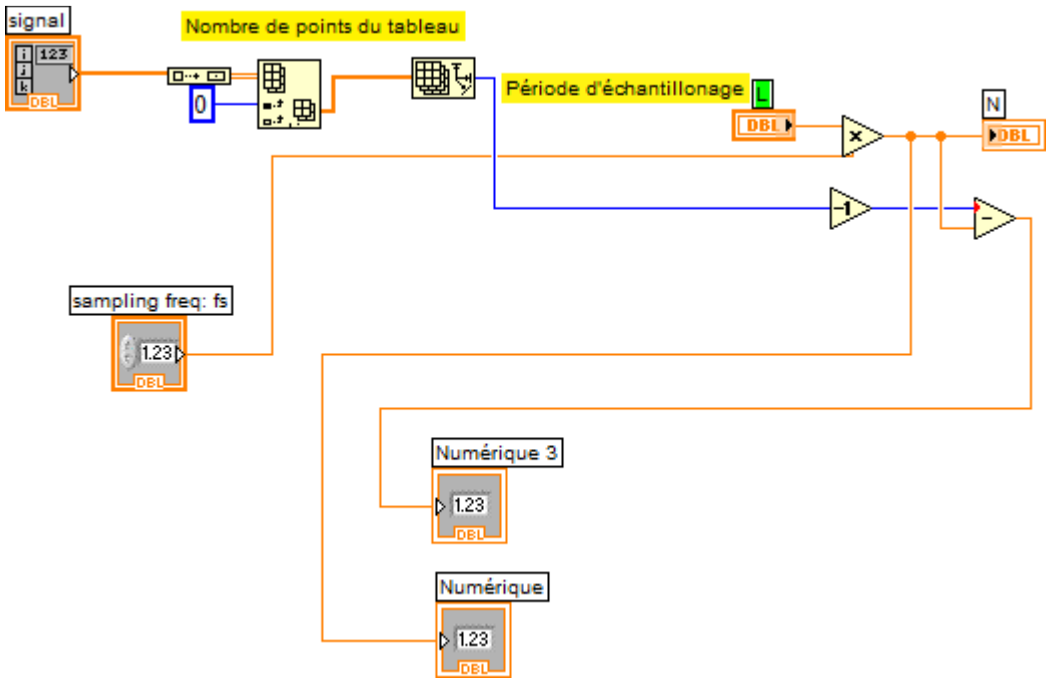
2.1 VI d'extraction des échantillons EMG à partir d'un fichier texte



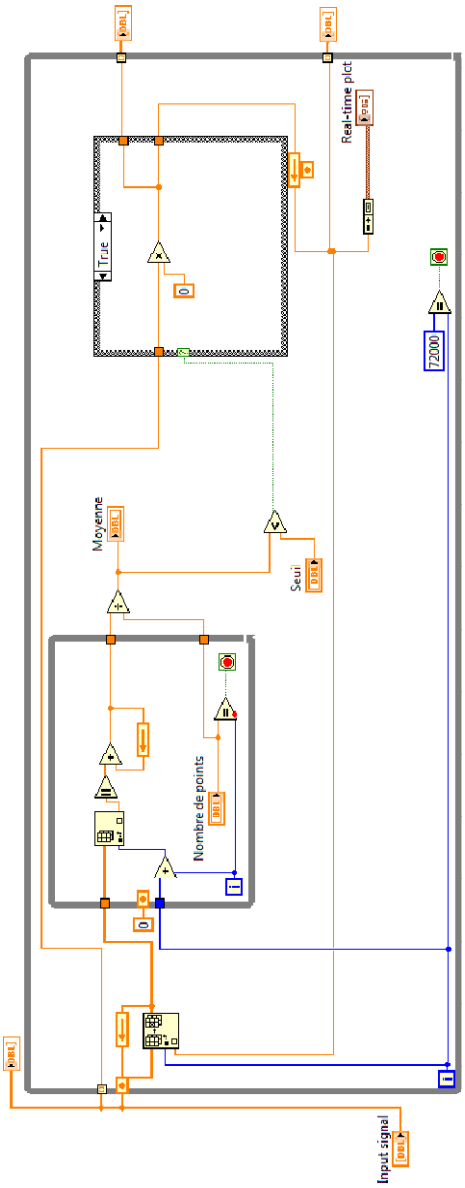
2.2 VI de l'algorithme SC



2.3 VI de calcul du nombre d'échantillons EMG

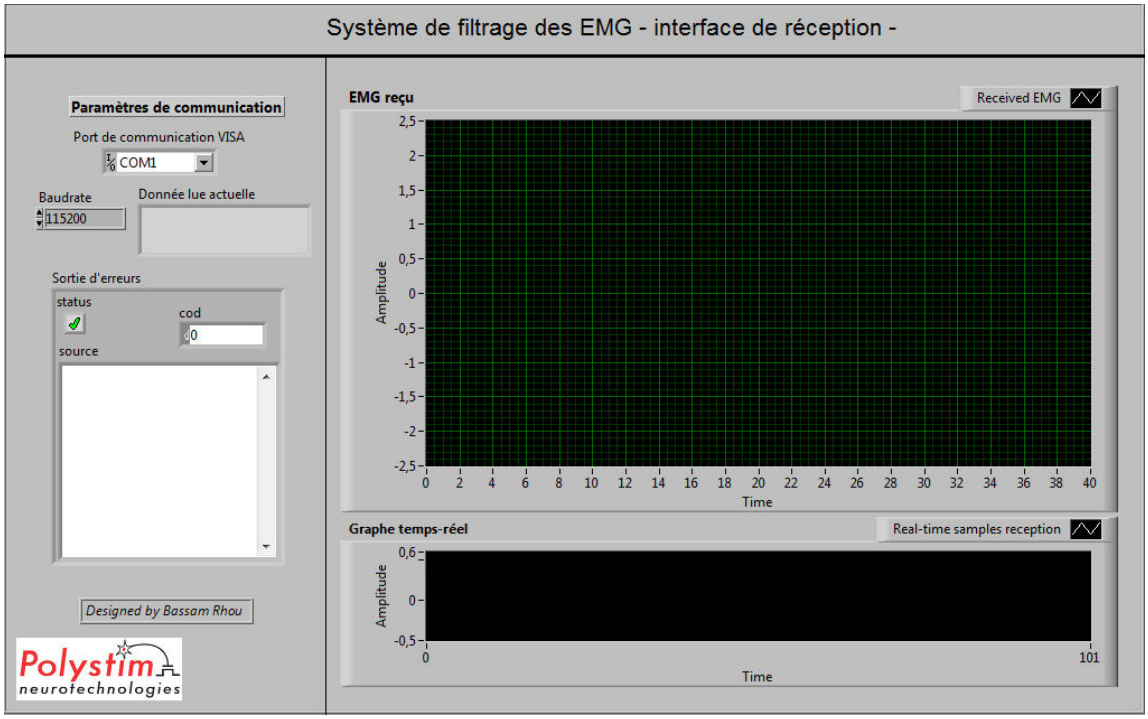


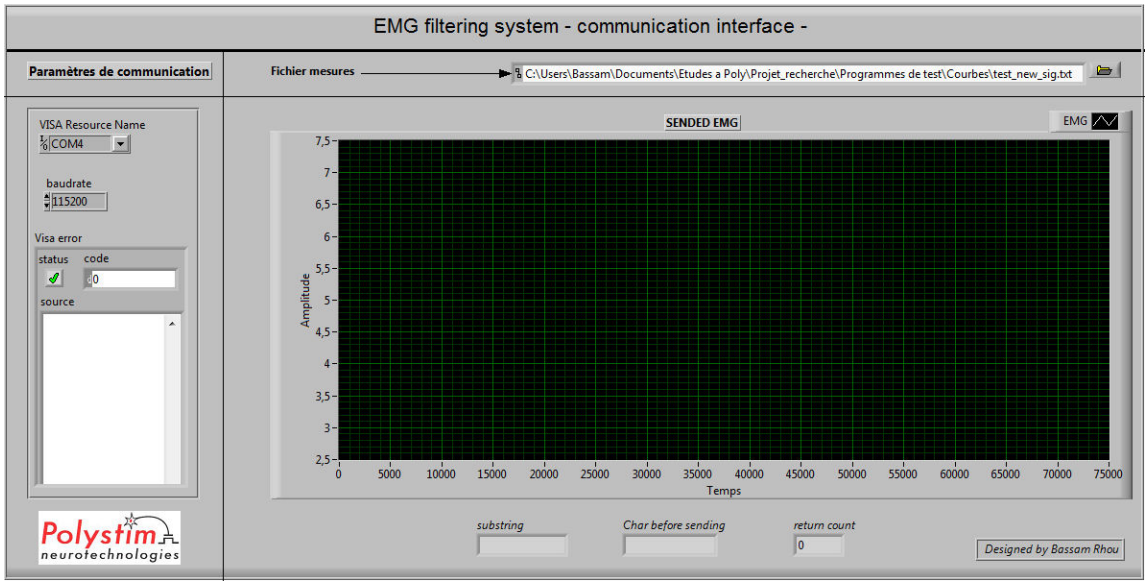
2.4 VI de l'algorithme d'élimination des ECG durant les expirations



III-Interfaces de communication

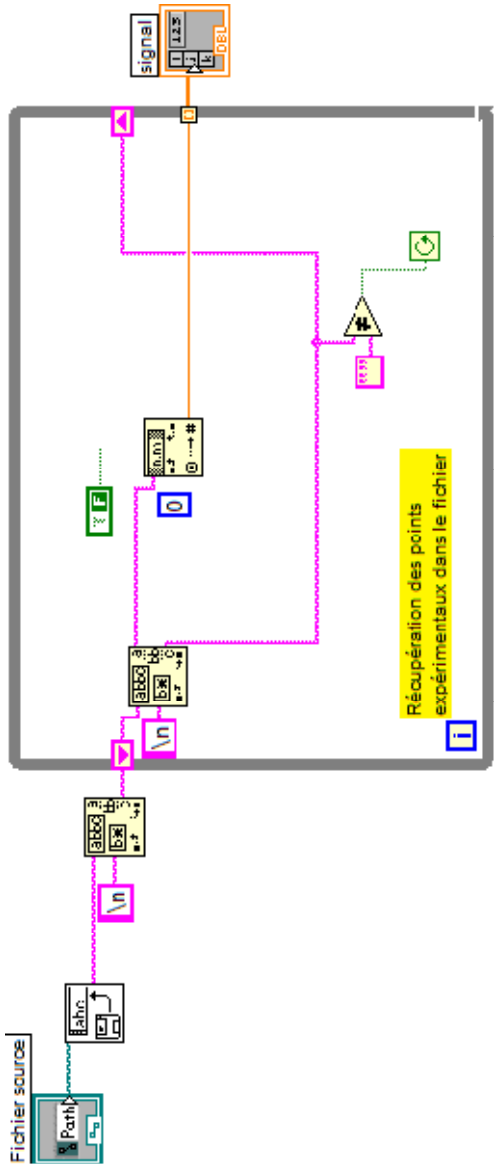
1. Onglets de réception et de transmission



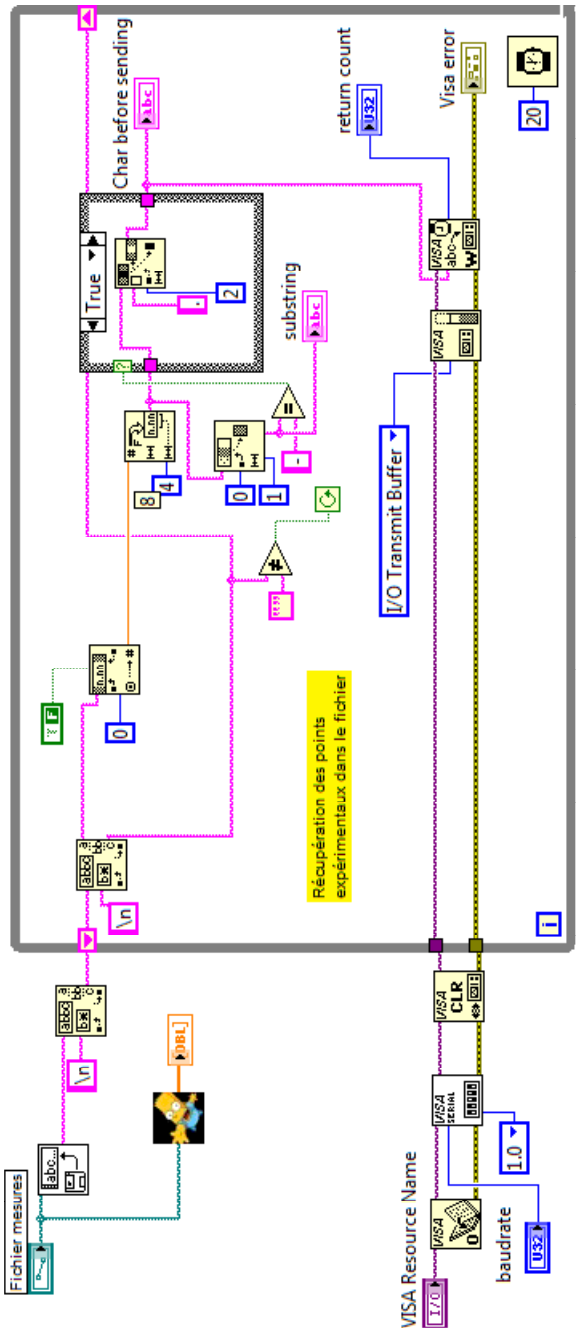


2. VI de l'envoi des échantillons EMG au système à microcontrôleurs

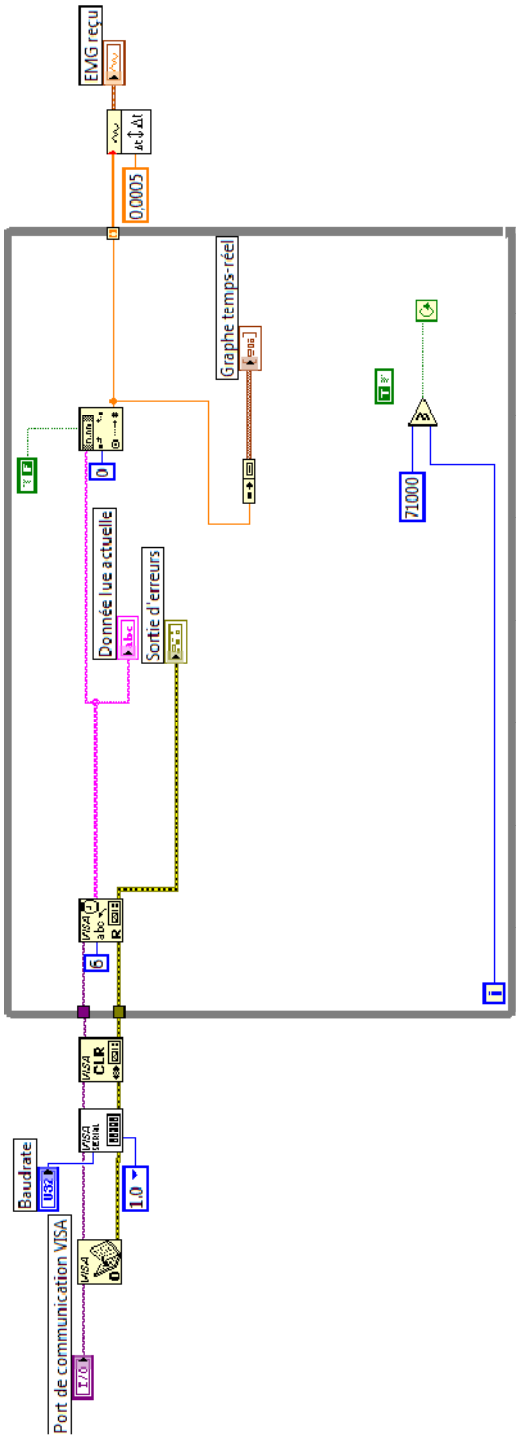
2.1 VI de l'extraction des données à envoyer



2.2 VI d'envoi des échantillons EMG extraits



3. VI de réception des échantillons EMG traités du système à microcontrôleurs



ANNEXE D – Carte de développement STK 500 d'ATMEL

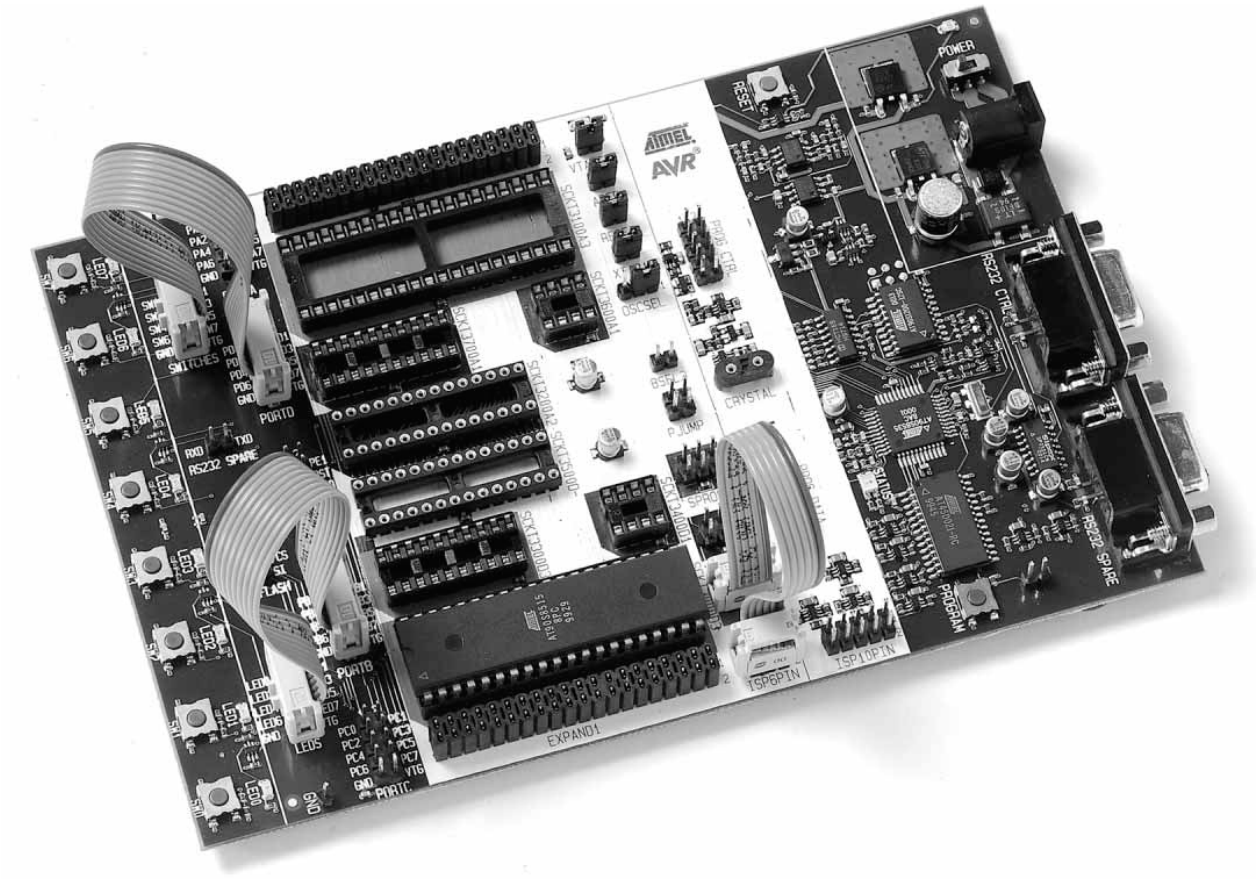


Figure D.1 : Carte de développement STK500 d'ATMEL (photo tirée de www.atmel.com)

ANNEXE E – Code source C et assembleur

I- Programme du microcontrôleur maître

```
////////////////////////////////////////////////////////////
//TITRE          : Systeme de filtrage des signaux EMGdi
//
//DATE                    : 10 Decembre 2007
//
//REVISION    : 3 Mars 2008
//
//MICROPROCESSEUR      : ATMEGA16
//
//AUTEUR       : Bassam Rhou
//
//DESCRITPION : programme du microcontrôleur maître
//
////////////////////////////////////////////////////////////


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/iom16.h>
#include <inttypes.h>

#define TAILLE 200
#define g 2.00000000          //Gain de l'algorithme SC
```

```
#define NPOLES 4
#define GAIN 1.131103948

void USART_init(unsigned int baud);
void USART_Transmit(unsigned int data);
unsigned char USART_Receive(void);
void SPI_MasterInit(void);
void SPI_MasterTransmit(char cData);
void ATTENTE(unsigned long int temps);
double sign(double f);

unsigned char sreg;
char mesure[10];
double results;
double ARV = 0;
double y = 0;                   //sortie du filtre passe-haut de BW
unsigned int i = 0;
unsigned int j = 0;
unsigned long int k = 0;
unsigned long int m = 0;
int n = 0;
int result_int;

char tempo;
//On définit xv et yv comme un type static pour pouvoir garder leurs valeurs pour la prochaine
      interruption
static double xv[5];
static double yv[5];    //Pour BW
double res_int[TAILLE];
double temp;
double temp2;
double temp3;
int indic_first = 0;            //est mis à 1 après la lecture des données initiales
```

```
unsigned long int a;
int sortir1 = 1;
double test_m;

//Interruption INT2
SIGNAL(SIG_INTERRUPT2)
{
sreg = SREG;
while((k < TAILLE) && (indic_first == 0))
        {
                while(n < 6)
                        {
                                mesure[n] = USART_Receive();
                                n++;
                        }
                n = 0;
                if(mesure[0] == '-')
                        tempo = USART_Receive();
                results = atof(mesure);
                results = results - 5.05;
                res_int[k] = results;
                ARV = ARV + fabs(results);
                temp  = results;
                xv[0] = xv[1];
                xv[1] = xv[2];
                xv[2] = xv[3];
                xv[3] = xv[4];
                xv[4] = temp / GAIN;
                yv[0] = yv[1];
                yv[1] = yv[2];
                yv[2] = yv[3];
                yv[3] = yv[4];
                yv[4] =   (xv[0] + xv[4]) - 4 * (xv[1] + xv[3]) + 6 * xv[2]
```

```
                        + ( -0.7816187403 * yv[0]) + (  3.3189386048 * yv[1])
                        + ( -5.2911525842 * yv[2]) + (  3.7537627567 * yv[3]);
            results = yv[4];

            sprintf(mesure,"%6.5f",results);       //A REMPLACER PAR 10.9

            while(n < 6)    //REMPLACER 7 PAR 10
                {
                    ATTENTE(5);
                    SPI_MasterTransmit(mesure[n]);
                    n++;
                }
            n = 0;
            m = 0;
            k++;
            sortir1 = 1;
            break;
        }

if(sortir1 == 0)
{
        n = 0;
        while(n < 6)            //REMPLACER 7 PAR 10
            {
                mesure[n] = USART_Receive();
                n++;
            }
        n = 0;
        if(mesure[0] == '-')
            tempo = USART_Receive();
        k = 0;
        indic_first = 1;
        ARV = ARV / TAILLE;
```

```
temp = ARV * g;
results = atof(mesure);
results = results - 5.05;
temp2 = fabs(results);
temp3 = temp2 - temp;
if (temp3 > 0)
        {
                results = sign(results);
                results = results * g;
                results = results * ARV;
        }
else
        results = results;
test_m = res_int[m];
ARV = ARV * TAILLE;
ARV = ARV - fabs(test_m);
ARV = ARV + fabs(results);
res_int[m] = results;
xv[0] = xv[1];
xv[1] = xv[2];
xv[2] = xv[3];
xv[3] = xv[4];
xv[4] = results / GAIN;
yv[0] = yv[1];
yv[1] = yv[2];
yv[2] = yv[3];
yv[3] = yv[4];
yv[4] =   (xv[0] + xv[4]) - 4 * (xv[1] + xv[3]) + 6 * xv[2]
    + ( -0.7816187403 * yv[0]) + (  3.3189386048 * yv[1])
    + ( -5.2911525842 * yv[2]) + (  3.7537627567 * yv[3]);
results = yv[4];
sprintf(mesure,"%6.5f",results);
n = 0;
```

```
        while(n < 6)
                {
                        ATTENTE(5);
                        SPI_MasterTransmit(mesure[n]);
                        n++;
                }
        n = 0;
        if(m == (TAILLE - 1))
                m = 0;
        else
                m++;
}

else
        {
                if(k == TAILLE)
                        {
                                sortir1 = 0;
                                k = 0;
                                indic_first = 1;
                        }
                else
                        sortir1 = 1;
        }
SREG = sreg;
}

//Programme principal
int main(void)
{
//Initialisation des ports
DDRD = 0x00;
DDRB = 0xB2;
```

```
//Initialisation USART: baudrate = 115200 baud
USART_init(1);

//Initialisation du CAN
//adc_init();

//Initialisation de la SPI
SPI_MasterInit();

//Initialisation de INT2
MCUCSR      |= 0x40;
 GICR           |= 0x20;
SREG           |= 0x80;
sei();

//Initialisation des constantes
i = 0;
j = 0;
k = 0;
m = 0;


while(1)
        {
                //Attente d'interruption
        }
return 1;
}

//////////////////////////////////////////////////////
/**********INITIALISATION USART***********/
//////////////////////////////////////////////////////
```

```
void USART_init(unsigned int baud)
{
//Set baud rate
UBRRH = (unsigned char)(baud>>8);
UBRRL = (unsigned char) baud;
//Enable receiver and transmitter
UCSRB = (1<<RXEN) | (1<<TXEN);
//Set frame format: 8data, 2stop bit
UCSRC = (1<<URSEL) | (1<<USBS) | (3<<UCSZ0);
}

//////////////////////////////////////////////////////
/*******ENVOI D'UN CHAR PORT SERIE********/
//////////////////////////////////////////////////////
void USART_Transmit(unsigned int data)
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<UDRE)) );
/* Put data into buffer, sends the data */
UDR = data;
}

//////////////////////////////////////////////////////
/*******RECOIS UN CHAR PORT SERIE*********/
//////////////////////////////////////////////////////
unsigned char USART_Receive( void )
{
/* Wait for data to be received */
while ( !(UCSRA & (1<<RXC)) );
/* Get and return received data from buffer */
return UDR;
}
```

```
//////////////////////////////////////////////////////
/*************Init du Master SPI************/
//////////////////////////////////////////////////////
void SPI_MasterInit(void)
{
/* Enable SPI, Master, set clock rate fck/16 */
SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPI2X)|(1<<SPR0);
PORTB &= 0xEF;                //PORTB4 = 0 => SS = 0
}

//////////////////////////////////////////////////////
/************Init de la transmission***********/
//////////////////////////////////////////////////////
void SPI_MasterTransmit(char cData)
{
/* Start transmission */
SPDR = cData;
/* Wait for transmission complete */
while(!(SPSR & (1<<SPIF)));
}

//////////////////////////////////////////////////////
/**************Fonction d'attente*********************/
//////////////////////////////////////////////////////
void ATTENTE(unsigned long int temps)
{
int i;
for(i=0;i<temps;i++)
        {
        }
}

//////////////////////////////////////////////////////
```

```
/***********Fonction de signe*************/
//////////////////////////////////////////////////////
double sign(double f)
{
double a;
if(f > 0)
        a = 1.00000000;
else
        {
                if(f < 0)
                        a = -1.00000000;
                else
                        a = 0.00000000;
        }
return a;
}
```

II- Programme du microcontrôleur esclave

```
;*****************************************************************
;TITRE: Tests: Filtrage des EMGdi – microcontroleur esclave(ATMEGA8515)
;DATE: 8 Février 2008
;REVISION:
;MICROPROCESSEUR: MEGA8515
;*****************************************************************


.include "m8515def.inc"


;*****************************************************************
;                   Interruption vector
```

```
;*****************************************************************
.org 0x0000
rjmp RESET                      /*Reset Handler*/
        nop
reti                            /*IRQ0 Handler*/
        nop
reti                            /*IRQ1 Handler*/
        nop
reti                            /*Timer2 Compare Handler*/
        nop
reti                            /*Timer2 Overflow Handler*/
        nop
reti                            /*Timer1 Capture Handler*/        nop
reti                            /*Timer1 CompareA Handler*/
        nop
reti                            /*Timer1 CompareB Handler*/
        nop
reti                            /*Timer1 Overflow Handler*/
        nop
reti                            /*Timer0 Overflow Handler*/
        nop
reti                            /*SPI Transfer Complete Handler*/
         nop
reti                            /*USART RX Complete Handler*/     nop
reti                            /*UDR Empty Handler*/
        nop
reti                            /*USART TX Complete Handler*/     nop
reti                            /*ADC Conversion Complete Handler*/ nop
reti                            /*EEPROM Ready Handler*/
        nop
reti                            /*Analog Comparator Handler*/     nop
reti                            /*Two-wire Serial Interface Handler*/
         nop
rjmp RESET                      /*IRQ2 Handler*/
        nop
reti                            /*Timer0 Compare Handler*/
        nop
```

```
reti                                    /*Store Program Memory Ready Handler*/
        nop

;*****************************************************************
;                   Définitions
;*****************************************************************

.def STATUS         = r18
.def tmp            = r19
.def cpt            = r20
.def test           = r23
;De 26 a 31, on a les registres X, Y, Z
.def pLOADL         = r28               ;pointeur de lecture de la RAM--Y
.def pLOADH         = r29               ;pointeur de lecture de la RAM--Y
.def pRAML          = r30                ;pointeur d'ecriture dans la RAM--Z
.def pRAMH          = r31               ;pointeur d'ecriture dans la RAM--Z

.equ SLAVE_READY = 3

;*****************************************************************
;                       RESET ROUTINE
;*****************************************************************
RESET:
        ldi r16,HIGH(RAMEND)
        out SPH,r16
        ldi r16,low(RAMEND)
        out SPL,r16

;*****************************************************************
;               Programme principal
;*****************************************************************

MAIN:
```

```
        clr     cpt
;Initialisation des ports
        ldi     r17,(1<<DDB6) | (1<<DDB3)
        out     DDRB,r17
        cbi     PORTB,3
        ldi     r17,0x00
        out     DDRE,r17

;Initialisation des constantes
        clr     cpt

;Initialisation des pointeurs sur la 1ere adresse de la RAM
        rcall   RAM_BEGIN_WR
        rcall   RAM_BEGIN_RD

;Initialisation de INT2
        in      r16, MCUCSR
        ori     r16, 0b01000000
        out     MCUCSR, r16
        in      r16, GICR
        ori     r16, 0b00100000
        out     GICR, r16
        rcall   SPI_SlaveInit
        rcall   USART_Init              ;initialisation USART

;Requete pour reception
        sbi             PORTB,SLAVE_READY
        nop
        nop
        nop
        nop
        nop
        nop
```

```
        nop
        cbi             PORTB,SLAVE_READY

ATTENTE1:
        inc     cpt
        cpi     cpt,0x30
        brne    ATTENTE1
        clr     cpt

BOUCLE:
//Reception et envoi 7 fois avant l'activation de l'interruption du Master
    rcall       SPI_SlaveReceive    ;réception des données envoyées par le ATMEGA16
        nop
        rcall   USART_Transmit      ;Transmission de ces données par liaison série
        inc     cpt
        cpi     cpt,0x07
        ;Si 7eme itération, activation de l'interruption du Master
        brne    BOUCLE
        clr     cpt
        sbi     PORTB,SLAVE_READY
        nop
        nop
        nop
        nop
        nop
        nop
        cbi     PORTB,SLAVE_READY
        nop
        rjmp    BOUCLE

;*********************************************************************
;                   Fonctions
;*********************************************************************
```

```
;**********************Initialisation USART**************************

USART_Init:
        ;init du USART
        ldi r16,0x01
        ldi r17,0
        out UBRRH,r17
        out UBRRL,r16
        ldi tmp,(1<<TXEN)|(1<<RXEN)
        out UCSRB,tmp
        ldi tmp,(1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1)
        out UCSRC,tmp
ret

;**********************Transmission de donnees************************
USART_Transmit:
        ; Wait for empty transmit buffer
        sbis UCSRA,UDRE
        rjmp USART_Transmit
        ; Put data (r16) into buffer, sends the data
        out UDR,r16
ret
;*********************************************************************

;**********************Initialisation du SPI**************************
SPI_SlaveInit:
        ; Set MISO output, all others input
        ;ldi r17,(1<<DDB6)
        ;out DDRB,r17
        ; Enable SPI
        ldi r17,(1<<SPE)
        out SPCR,r17
```

```
ret

;***********Réceptions des donnees du ATMEGA16 par SPI*****************
SPI_SlaveReceive:
        ; Wait for reception complete
        sbis    SPSR,SPIF
        rjmp    SPI_SlaveReceive
        in test,SPSR
        in r16,SPDR
ret
;*********************************************************************

;*********************************************************************
;Fonction qui pointe au début de la RAM (ECRITURE)
;*********************************************************************
RAM_BEGIN_WR:
        ldi     pRAMH,0x00
        ldi     pRAML,0x60
ret
;*********************************************************************

;*********************************************************************
;Fonction qui pointe au début de la RAM (LECTURE)
;*********************************************************************
RAM_BEGIN_RD:
        ldi     pLOADH,0x00
        ldi     pLOADL,0x60
ret
;*********************************************************************
```

ANNEXE F – Code VHDL implémentant l'algorithme SC

```
--============================================================
-- TITLE : cellule du diviseur
-- DESCRIPTION : test
-- FILE : cellule.vhd
--============================================================
-- CREATION
-- DATE AUTHOR PROJECT REVISION COMMENTS
-- 2009/10/04 Bassam Rhou, jr. eng.
--============================================================
-- MODIFICATION HISTORY
-- DATE AUTHOR PROJECT REVISION COMMENTS
--============================================================
-- Copyright (c) 2008
--============================================================
        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_unsigned.all;
        use work.all;

        entity cellule is  -- Controlled Add/Subtract cell
          port (
            divisor      : in  std_logic;
            T           : in  std_logic;
            r_in         : in  std_logic;
            cin          : in  std_logic;
            r_out        : out std_logic;
            cout         : out std_logic);
        end entity cellule;

        architecture circuits of cellule is
          signal tt : std_logic;
        begin
          tt          <= T xor divisor after 10 ps;
          r_out <= tt xor r_in xor cin after 10 ps;
          cout         <= (tt and r_in) or (tt and cin) or
                          (r_in and cin) after 10 ps;
end architecture circuits;
```

```
--=====================================================================
        -- TITLE : Diviseur
        -- DESCRIPTION : Diviseur 2N bits par 9 bits
        -- FILE : div_2N.vhd
--=====================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2009/10/04 Bassam Rhou, jr. eng.
--=====================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--=====================================================================
        -- Copyright (c) 2008
--=====================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_unsigned.all;
        use work.all;

        entity div_N is
        generic (N: INTEGER:=12);
          port (
            ARV_in    : in  std_logic_vector(2*N downto 0);
            dvr       : in  std_logic_vector(8 downto 0);
            ARV_out   : out std_logic_vector(N-1 downto 0);
            r         : out std_logic_vector(8 downto 0)
          );
        end entity div_N;

        architecture circuit_diviseur of div_N is
          signal T  : std_logic_vector(16 downto 0);
          signal c  : std_logic_vector(152 downto 0);
          signal rr : std_logic_vector(152 downto 0);

        begin
        --Mettre première ligne supérieure du schéma
        T(16) <= '1' after 10 ps;
           GEN1: for i in 8 downto 1 generate
```

```
  cellule: entity WORK.cellule port map(
  dvr(i), T(16), ARV_in(16 + i), c(144 -1 + i), rr(144 + i), c(144 + i));
end generate;
cellule1: entity WORK.cellule port map(
dvr(0), T(16), ARV_in(16), T(16), rr(144), c(144));
T(15) <= not rr(152) after 10 ps;
  --GEN2: for j in 16 downto 0 generate
    D2: for i in 8 downto 1 generate
      cellule: entity WORK.cellule port map(
        dvr(i), T(15), rr(144 - 1 + i), c(135 - 1 + i), rr(135 + i), c(135 + i));
    end generate;
    cellule2: entity WORK.cellule port map(
    dvr(0), T(15), ARV_in(15), T(15), rr(135), c(135));
  --end generate;

T(14) <= not rr(143) after 10 ps;
  --GEN3:for j in 16 downto 0 generate
    D3: for i in 8 downto 1 generate
      cellule: entity WORK.cellule port map(
        dvr(i), T(14), rr(135 - 1 + i), c(126 - 1 + i), rr(126 + i), c(126 + i));
    end generate;
    cellule3: entity WORK.cellule port map(
    dvr(0), T(14), ARV_in(14), T(14), rr(126), c(126));
  --end generate;

T(13) <= not rr(134) after 10 ps;
  --GEN4: for j in 16 downto 0 generate
    D4: for i in 8 downto 1 generate
      cellule: entity WORK.cellule port map(
        dvr(i), T(13), rr(126 - 1 + i), c(117 - 1 + i), rr(117 + i), c(117 + i));
    end generate;
    cellule4: entity WORK.cellule port map(
    dvr(0), T(13), ARV_in(13), T(13), rr(117), c(117));
  --end generate;

T(12) <= not rr(125) after 10 ps;
  --GEN5: for j in 16 downto 0 generate
    D5: for i in 8 downto 1 generate
      cellule: entity WORK.cellule port map(
        dvr(i), T(12), rr(117 - 1 + i), c(108 - 1 + i), rr(108 + i), c(108 + i));
    end generate;
    cellule5: entity WORK.cellule port map(
    dvr(0), T(12), ARV_in(12), T(12), rr(108), c(108));
  --end generate;
```

```
T(11) <= not rr(116) after 10 ps;
  --GEN6: for j in 16 downto 0 generate
    D6: for i in 8 downto 1 generate
      cellule: entity WORK.cellule port map(
        dvr(i), T(11), rr(108 - 1 + i), c(99 - 1 + i), rr(99 + i), c(99 + i));
    end generate;
    cellule6: entity WORK.cellule port map(
    dvr(0), T(11), ARV_in(11), T(11), rr(99), c(99));
  --end generate;

T(10) <= not rr(107) after 10 ps;
  --GEN7: for j in 16 downto 0 generate
    D7: for i in 8 downto 1 generate
      cellule: entity WORK.cellule port map(
        dvr(i), T(10), rr(99 - 1 + i), c(90 - 1 + i), rr(90 + i), c(90 + i));
    end generate;
    cellule7: entity WORK.cellule port map(
    dvr(0), T(10), ARV_in(10), T(10), rr(90), c(90));
  --end generate;

T(9) <= not rr(98) after 10 ps;
  --GEN8: for j in 16 downto 0 generate
    D8: for i in 8 downto 1 generate
      cellule: entity WORK.cellule port map(
        dvr(i), T(9), rr(90 - 1 + i), c(81 - 1 + i), rr(81 + i), c(81 + i));
    end generate;
    cellule8: entity WORK.cellule port map(
    dvr(0), T(9), ARV_in(9), T(9), rr(81), c(81));
  --end generate;

T(8) <= not rr(89) after 10 ps;
  --GEN9: for j in 16 downto 0 generate
    D9: for i in 8 downto 1 generate
      cellule: entity WORK.cellule port map(
        dvr(i), T(8), rr(81 - 1 + i), c(72 - 1 + i), rr(72 + i), c(72 + i));
    end generate;
    cellule9: entity WORK.cellule port map(
    dvr(0), T(8), ARV_in(8), T(8), rr(72), c(72));
  --end generate;

T(7) <= not rr(80) after 10 ps;
  --GEN10: for j in 16 downto 0 generate
    D10: for i in 8 downto 1 generate
```

```
        cellule: entity WORK.cellule port map(
          dvr(i), T(7), rr(72 - 1 + i), c(63 - 1 + i), rr(63 + i), c(63 + i));
      end generate;
      cellule10: entity WORK.cellule port map(
      dvr(0), T(7), ARV_in(7), T(7), rr(63), c(63));
    --end generate;

T(6) <= not rr(71) after 10 ps;
    --GEN11: for j in 16 downto 0 generate
      D11: for i in 8 downto 1 generate
        cellule: entity WORK.cellule port map(
          dvr(i), T(6), rr(63 - 1 + i), c(54 - 1 + i), rr(54 + i), c(54 + i));
      end generate;
      cellule11: entity WORK.cellule port map(
      dvr(0), T(6), ARV_in(6), T(6), rr(54), c(54));
    --end generate;

T(5) <= not rr(62) after 10 ps;
    --GEN12: for j in 16 downto 0 generate
      D12: for i in 8 downto 1 generate
        cellule: entity WORK.cellule port map(
          dvr(i), T(5), rr(54 - 1 + i), c(45 - 1 + i), rr(45 + i), c(45 + i));
      end generate;
      cellule12: entity WORK.cellule port map(
      dvr(0), T(5), ARV_in(5), T(5), rr(45), c(45));
    --end generate;

T(4) <= not rr(53) after 10 ps;
    --GEN13: for j in 16 downto 0 generate
      D13: for i in 8 downto 1 generate
        cellule: entity WORK.cellule port map(
          dvr(i), T(4), rr(45 - 1 + i), c(36 - 1 + i), rr(36 + i), c(36 + i));
      end generate;
      cellule13: entity WORK.cellule port map(
      dvr(0), T(4), ARV_in(4), T(4), rr(36), c(36));
    --end generate;

T(3) <= not rr(44) after 10 ps;
    --GEN14: for j in 16 downto 0 generate
      D14: for i in 8 downto 1 generate
        cellule: entity WORK.cellule port map(
          dvr(i), T(3), rr(36 - 1 + i), c(27 - 1 + i), rr(27 + i), c(27 + i));
      end generate;
      cellule14: entity WORK.cellule port map(
```

```
      dvr(0), T(3), ARV_in(3), T(3), rr(27), c(27));
    --end generate;

  T(2) <= not rr(35) after 10 ps;
    --GEN15: for j in 16 downto 0 generate
      D15: for i in 8 downto 1 generate
        cellule: entity WORK.cellule port map(
          dvr(i), T(2), rr(27 - 1 + i), c(18 - 1 + i), rr(18 + i), c(18 + i));
      end generate;
      cellule15: entity WORK.cellule port map(
      dvr(0), T(2), ARV_in(2), T(2), rr(18), c(18));
    --end generate;

  T(1) <= not rr(26) after 10 ps;
    --GEN16: for j in 16 downto 0 generate
      D16: for i in 8 downto 1 generate
        cellule: entity WORK.cellule port map(
          dvr(i), T(1), rr(18 - 1 + i), c(9 - 1 + i), rr(9 + i), c(9 + i));
      end generate;
      cellule16: entity WORK.cellule port map(
      dvr(0), T(1), ARV_in(1), T(1), rr(9), c(9));
    --end generate;

--Mettre dernière ligne inférieure du schéma
  T(0) <= not rr(17) after 10 ps;
  GEN: for i in 8 downto 1 generate
    cellule: entity WORK.cellule port map(
    dvr(i), T(0), rr(9 - 1 + i), c(i-1), rr(i), c(i));
  end generate;
  cellule17: entity WORK.cellule port map(
  dvr(0), T(0), ARV_in(0), T(0), rr(0), c(0));
  ARV_out(11) <= T(10);
  ARV_out(10) <= T(9);
  ARV_out(9)  <= T(8);
  ARV_out(8)  <= T(7);
  ARV_out(7)  <= T(6);
  ARV_out(6)  <= T(5);
  ARV_out(5)  <= T(4);
  ARV_out(4)  <= T(3);
  ARV_out(3)  <= T(2);
  ARV_out(2)  <= T(1);
  ARV_out(1)  <= T(0);
  ARV_out(0)  <= not rr(8) after 10 ps;
  r(8) <= rr(8);
```

```
    r(7) <= rr(7);
    r(6) <= rr(6);
    r(5) <= rr(5);
    r(4) <= rr(4);
    r(3) <= rr(3);
    r(2) <= rr(2);
    r(1) <= rr(1);
    r(0) <= rr(0);
end architecture circuit_diviseur;
```

```
--=============================================================================
    -- TITLE : enable
    -- DESCRIPTION : S'active lorsque le compteur atteint sa valeur limite
    -- FILE : enable.vhd
--=============================================================================
    -- CREATION
    -- DATE AUTHOR PROJECT REVISION COMMENTS
    -- 2008/08/21 Bassam Rhou, jr. eng.
--=============================================================================
    -- MODIFICATION HISTORY
    -- DATE AUTHOR PROJECT REVISION COMMENTS
--=============================================================================
    -- Copyright (c) 2008
--=============================================================================

    library ieee;
    library work;
    use ieee.std_logic_misc.all;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_arith.all;
    use ieee.numeric_std.all;
    use ieee.std_logic_signed.all;
    use work.all;

    entity enable is
    generic (N: integer := 12);
    port (
        data_en_in    : in  std_logic_vector(N-1 downto 0);
```

```
            over_cpt      : in  std_logic;
            data_en_out    : out std_logic_vector(N-1 downto 0)
            );
        end enable;

        architecture behavior_enable of enable is
        begin
          process(over_cpt, data_en_in)
            begin
              if (over_cpt = '1') then
                data_en_out <= Data_en_in;
             end if;
          end process;
end behavior_enable;

--=============================================================================
        -- TITLE : enable24_24
        -- DESCRIPTION : S'active lorsque le compteur atteint sa valeur limite
        -- FILE : enable.vhd
--=============================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/21 Bassam Rhou, jr. eng.
--=============================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--=============================================================================
        -- Copyright (c) 2008
--=============================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;

        entity enable24_24 is
        generic (N: integer := 12);
        port (
            data_en24_in     : in  std_logic_vector(2*N downto 0);
            over_cpt24       : in  std_logic;
```

```
            data_en24_out    : out std_logic_vector(2*N downto 0)
            );
        end enable24_24;

        architecture behavior_enable24 of enable24_24 is
        begin
          process(over_cpt24, data_en24_in)
            begin
              if over_cpt24 = '1' then
                data_en24_out <= Data_en24_in;
              else
                data_en24_out <= "000000000000000000000000";
              end if;
          end process;
end behavior_enable24;

--=============================================================================
        -- TITLE : Valeur absolue
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : abs_value.vhd
--=============================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/21 Bassam Rhou, jr. eng.
--=============================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--=============================================================================
        -- Copyright (c) 2008
--=============================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;


        entity abs_value is
        generic (N: integer := 12);
        port (
```

```
            entr    : in  std_logic_vector(N-1 downto 0);
            absol   : out std_logic_vector(N-1 downto 0)
            );
        end abs_value;

        architecture behavior_abs of abs_value is
        begin
          process(entr)
            begin
              if entr(N-1) = '1' then
                absol(N-2 downto 0) <= not(entr(N-2 downto 0))+ "00000000001";   --Faire
attention au nombre à ajouter si on change N
                absol(N-1)        <= '0';
              else
                absol             <= entr;
             end if;
        --absol   <= conv_std_logic_vector(abs(conv_integer(entr)),12);
          end process;
end behavior_abs;


--================================================================================
        -- TITLE : Half adder
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : HALF_ADD.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/25 Bassam Rhou, jr. eng.
--================================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--================================================================================
        -- Copyright (c) 2008
--================================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;
```

```
        entity HALF_ADDER is
        port (
        a,b       : in  std_logic;
        s,cout    : out std_logic
        );
        end HALF_ADDER;

        architecture dataflow_hadd of HALF_ADDER is
        begin
          s   <= a xor b;
          cout <= a and b;
end dataflow_hadd;

--================================================================================
        -- TITLE : Signum multiplication
        -- DESCRIPTION : Component of the spike-clipping algorithm - Multiplate signum function and g*ARV
        -- FILE : Mult_sig.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/21 Bassam Rhou, jr. eng.
--================================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--================================================================================
-- Copyright (c) 2008
--================================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;

        entity Mult_sig is
        generic (N: integer := 12);
        port (
            Data_sig    : in  std_logic_vector(N-1 downto 0);
            Data        : in  std_logic_vector(N-1 downto 0);
```

```
            threshold    : out std_logic_vector(N-1 downto 0)
            );
        end Mult_sig;

        architecture behavior_mult_sig of Mult_sig is
        begin
          process(Data_sig,Data)
            begin
              if Data_sig(N-1) = '1' then
                threshold(N-2 downto 0) <= not(Data(N-2 downto 0)) + "00000000001";
                threshold(N-1)         <= '1';
              else
                threshold <= Data;
              end if;
          end process;
end behavior_mult_sig;

--================================================================================
        -- TITLE : Adder with enable
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : ADDNE.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/25 Bassam Rhou, jr. eng.
--================================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--================================================================================
        -- Copyright (c) 2008
--================================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;

        entity ADDNE is
        generic (N: INTEGER:=12);
        port (
```

```
        A,B     : in std_logic_vector(N-1 downto 0);
        S       : out std_logic_vector(N downto 0);
        ENABLE   : in std_logic
        );
        end;

        architecture structural_addne of ADDNE is
        signal CARRY: std_logic_vector (N-1 downto 0);
        signal AA: std_logic_vector(N-1 downto 0);
        component ADD1 port (A,B,CIN: in std_logic; S,COUT: out std_logic); end component;
        component HALF_ADDER port (a,b: in std_logic; s,cout: out std_logic); end component;
        -- configurations defaulted
        begin
        AA(0)<=A(0) and ENABLE;
        HA: HALF_ADDER port map(AA(0),B(0),S(0),CARRY(0));
        G1: for I in 1 to N-1 generate
           AA(I)<=A(I) and ENABLE;
           XI: ADD1 port map (AA(I),B(I),CARRY(I-1),S(I),CARRY(I));
        end generate;
        S(N)<=CARRY(N-1);
end structural_addne;

--================================================================================
        -- TITLE : Memory
        -- DESCRIPTION : Component of the spike-clipping algorithm -second part-
        -- FILE : mem_o.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/09/09 Bassam Rhou, jr. eng.
--================================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--================================================================================
        -- Copyright (c) 2008
--================================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
```

```
use work.all;


entity mem_o is
generic (width : integer := 200; N: integer := 12);
port (
     sk         : in std_logic_vector(N-1 downto 0);
     new_sk     : in std_logic_vector(N-1 downto 0);
     remplace_en : in std_logic;
     clk        : in std_logic;
     rst_n      : in  std_logic;
     oNm        : out std_logic_vector(N-1 downto 0);
     om         : out std_logic_vector(N-1 downto 0)
    );
end mem_o;

architecture behavior_mem_o of mem_o is
type reg_type is array (0 to width) of std_logic_vector(N-1 downto 0);
signal reg_int: reg_type;
signal compt  : std_logic_vector(9 downto 0);
begin
process(rst_n, clk, sk, new_sk, remplace_en)
  begin
  if(rst_n = '0') then
     oNm   <= "000000000000";
     om    <= "000000000000";
     compt <= "0000000000";
     for i in 0 to width loop
       reg_int(i) <= (others => '0');
     end loop;
  elsif(clk'event and clk = '1') then
     --Écrire la donnée d'entrée
     reg_int(conv_integer(compt)) <= sk;
     if remplace_en = '1' then
       reg_int(conv_integer(compt)) <= new_sk;
       oNm   <= new_sk;
     else
       oNm   <= sk;
     end if;
     --Envoyer les données de sortie
     if compt = width - 1 then
       om    <= reg_int(0);
       --oNm   <= sk;
       compt <= "0000000000";
```

```
            else
              om   <=  reg_int(conv_integer(compt) + 1);
              --oNm   <=  sk;
              compt <=  compt + 1;
            end if;
          end if;
        end process;
end behavior_mem_o;

--=============================================================================
        -- TITLE : Buffer à une entrée
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : Buff2.vhd
--=============================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/11/25 Bassam Rhou, jr. eng.
--=============================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--=============================================================================
        -- Copyright (c) 2008
--=============================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;


        entity Buff2 is
        generic (N: integer := 12);
        port (
             D_in       : in  std_logic_vector(N-1 downto 0);
             clk_in     : in  std_logic;
             rst_n      : in  std_logic;
             D_out       : out std_logic_vector(N-1 downto 0)
             );
        end Buff2;
```

```
        architecture behavior_buff2 of Buff2 is
        begin
          process(clk_in,rst_n)
            begin
              if(rst_n = '0') then
                 D_out <= "000000000000";
              elsif (clk_in'event and clk_in = '1') then    --voir s'il ne faut pas plutot mettre un
événement sur Data_in plutôt que sur clk
                 D_out <= D_in;
              end if;
           end process;
end behavior_buff2;



--================================================================================
        -- TITLE : 12 bits subtraction
        -- DESCRIPTION : Component of the spike-clipping algorithm -second part-
        -- FILE : Soustraction.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/09/09 Bassam Rhou, jr. eng.
--================================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--================================================================================
        -- Copyright (c) 2008
--================================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_unsigned.all;
        use work.all;

        entity Soustraction is
        generic (N: integer := 12);
        port (
             opA      : in std_logic_vector(N-1 downto 0);
             opB      : in std_logic_vector(N-1 downto 0);
```

```
            reset_n   : in std_logic;
            result    : out std_logic_vector(N-1 downto 0);
            sous_sign : out std_logic
            );
        end Soustraction;

        architecture behavior_soustraction of Soustraction is
        begin
        process(reset_n,opA,opB)
          begin
          if(reset_n = '0') then
            result <= (others => '0');
            sous_sign <= '0';
          elsif(opA >= opB) then
             result    <= (opA - opB);
             sous_sign <= '0';
          else
             result   <= (opB - opA);
             sous_sign <= '1';
          end if;
        end process;
end behavior_soustraction;

--================================================================================
        -- TITLE : 12 bits adder
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : ADDER_12.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/20 Bassam Rhou, jr. eng.
--================================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--================================================================================
        -- Copyright (c) 2008
--================================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
```

```
        use ieee.std_logic_signed.all;
        use work.all;

        entity ADDER1212 is
        generic (N: integer := 12);
        port (
             o1      : in std_logic_vector(2*N downto 0);
             o2      : in std_logic_vector(N-1 downto 0);
             add_sst  : in std_logic;
             reset_n  : in std_logic;
             result   : out std_logic_vector(2*N downto 0)
             );
        end ADDER1212;

        architecture behavior_ADDER1212 of ADDER1212 is
        begin
        process(reset_n,o1,o2,add_sst)
          begin
          if(reset_n = '0') then
            result <= "000000000000000000000000";
          else
            if add_sst = '1' then
              result   <= o1 - ("0000000000000" & o2);
            else
              result   <= o1 + ("0000000000000" & o2);
            end if;
          end if;
        end process;
        end behavior_ADDER1212;

--================================================================================
        -- TITLE : Select_ARV
        -- DESCRIPTION : Selects used ARV
        -- FILE : Select.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/10/20 Bassam Rhou, jr. eng.
--================================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--================================================================================
        -- Copyright (c) 2008
--================================================================================
```

```
        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_unsigned.all;
        use work.all;

        entity Select_ARV is
        generic (N: INTEGER:=12);
        port (
          ARV1        : in std_logic_vector(2*N downto 0);
          ARV         : in std_logic_vector(2*N downto 0);
          en_sel      : in std_logic;
          RST_N       : in std_logic;
          clk         : in std_logic;
          out_select   : out std_logic_vector(2*N downto 0)
        );
        end Select_ARV;

        architecture dataflow_Select of Select_ARV is
        begin
          process(clk,ARV1,RST_N,en_sel,ARV)
             begin
               if(RST_N = '0') then
                  out_select <= "000000000000000000000000";
               elsif(clk'event and clk = '1') then
                 if en_sel = '1' then
                    out_select <= ARV1;
                 else
                    out_select <= ARV;

                 end if;
               end if;
          end process;
end dataflow_Select;

--================================================================================
        -- TITLE : controleur de cas
        -- DESCRIPTION : détermine la sortie selon le résultat du comparateur
        -- FILE : ctrl_case.vhd
--================================================================================
```

```
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/22 Bassam Rhou, jr. eng.
--============================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--============================================================================
        -- Copyright (c) 2008
--============================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;


        entity ctrl_case is
        generic (N: integer := 12);
        port (
             x1       : in  std_logic_vector(N-1 downto 0);
             product   : in  std_logic_vector(N-1 downto 0);
             comp      : in std_logic;
             mod_x1    : out std_logic_vector(N-1 downto 0)
            );
        end ctrl_case;

        architecture behavior_ctrl_case of ctrl_case is
        begin
        process(comp,x1,product)
          begin
            if (comp = '1') then
              mod_x1 <= product;
            else
              mod_x1 <= x1;
            end if;
          end process;
end behavior_ctrl_case;

--============================================================================
        -- TITLE : Controleur initial
```

```
        -- DESCRIPTION : fait en sorte que la sortie = à l'entrée pendant le calcul de ARV(1)
        -- FILE : ctrl_init.vhd
--============================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/09/12 Bassam Rhou, jr. eng.
--============================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--============================================================================
        -- Copyright (c) 2008
--============================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;


        entity ctrl_init is
        generic (N: integer := 12);
        port (
             in_case       : in  std_logic_vector(N-1 downto 0);
             in_buf        : in  std_logic_vector(N-1 downto 0);
             in_cpt        : in  std_logic;
             final_res     : out std_logic_vector(N-1 downto 0)
             );
        end ctrl_init;

        architecture behavior_ctrl_init of ctrl_init is
        begin
          process(in_cpt,in_case,in_buf)
            begin
              if(in_cpt = '0') then
                final_res <= in_buf;
              else
                final_res <= in_case;
              end if;
          end process;
end behavior_ctrl_init;
```

```
--============================================================================
        -- TITLE : adder 1 bit
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : ADD1.vhd
--============================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/25 Bassam Rhou, jr. eng.
--============================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--============================================================================
        -- Copyright (c) 2008
--============================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;

        entity ADD1 is
        port(
        A, B, CIN : in std_logic;
        S, COUT : out std_logic
        );
        end ADD1;

        architecture DATAFLOW of ADD1 is
        signal INTER : std_logic;
        Begin
          INTER <= A xor B ;
          S <= INTER xor CIN;
          COUT<=(A and B)or(INTER and CIN);
end DATAFLOW;

--============================================================================
        -- TITLE : 12 bits/25 bits adder
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : ADDER_12.vhd
```

```
--=============================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/20 Bassam Rhou, jr. eng.
--=============================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--=============================================================================
        -- Copyright (c) 2008
--=============================================================================
       library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_unsigned.all;
        use work.all;

        entity ADDER_12 is
        generic (N: integer := 12);
        port (
             op1     : in std_logic_vector(N-1 downto 0);
             op2     : in std_logic_vector(2*N downto 0);
             reset_n  : in std_logic;
             result   : out std_logic_vector(2*N downto 0)
             );
        end ADDER_12;

        architecture behavior_ADDER_12 of ADDER_12 is
        begin
        process(reset_n,op1,op2)
          begin
          if(reset_n = '0') then
            result <= "000000000000000000000000";
          else
            result   <= "0000000000000"&op1 + op2;
          end if;
        end process;
end behavior_ADDER_12;

--=============================================================================
        -- TITLE : Buffer
        -- DESCRIPTION : Component of the spike-clipping algorithm
```

```
        -- FILE : Buff.vhd
--============================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/21 Bassam Rhou, jr. eng.
--============================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--============================================================================
        -- Copyright (c) 2008
--============================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;

        entity Buff is
        generic (N: integer := 12);
        port (
             Data_in     : in  std_logic_vector(N-1 downto 0);
             Data_add    : in  std_logic_vector(2*N downto 0);
             clk_in      : in  std_logic;
             rst_n       : in  std_logic;
             Data1_out   : out std_logic_vector(N-1 downto 0);
             Data2_out   : out std_logic_vector(2*N downto 0)
             );
        end Buff;

        architecture behavior_buff of Buff is
        begin
          process(clk_in,rst_n)
            begin
             if(rst_n = '0') then
                Data1_out <= "000000000000";
                Data2_out <= "000000000000000000000000";
             elsif (clk_in'event and clk_in = '1') then    --voir s'il ne faut pas plutot mettre un
événement sur Data_in plutôt que sur clk
                Data1_out <= Data_in;
                Data2_out <= Data_add;
```

```
            end if;
        end process;
end behavior_buff;

--=============================================================================
        -- TITLE : comparateur
        -- DESCRIPTION : compare les deux entrées
        -- FILE : Comparateur.vhd
--=============================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/22 Bassam Rhou, jr. eng.
--=============================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--=============================================================================
        -- Copyright (c) 2008
--=============================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_unsigned.all;
        use work.all;

        entity comparateur is
        generic (N: integer := 12);
        port (
             dt_1     : in  std_logic_vector(N-1 downto 0);
             dt_2     : in  std_logic_vector(N-1 downto 0);
             sup_inf   : out std_logic
            );
        end comparateur;

        architecture behavior_comparateur of comparateur is
        begin
        process(dt_1,dt_2)
          begin
            if ("0" & dt_1 > "0" & dt_2) then
              sup_inf <= '1';
            else
```

```
            sup_inf <= '0';
          end if;
        end process;
end behavior_comparateur;

--================================================================================
        -- TITLE : Multiplicateur
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : multiplicateur.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/25 Bassam Rhou, jr. eng.
--================================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--================================================================================
        -- Copyright (c) 2008
--================================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_unsigned.all;
        use work.all;

        entity multiplicateur is
        generic (N:INTEGER:=12);
        port (
        A,B: in std_logic_vector(N-1 downto 0);
        --S: out std_logic_vector(2*N-1 downto 0)
        S: out std_logic_vector(N-1 downto 0)
        );
        end multiplicateur;

        architecture STRUCT of multiplicateur is
        type ADHOC is array(N downto 0) of std_logic_vector(N downto 0);
        signal CARRY:ADHOC;
        signal tmp_s : std_logic_vector(2*N-1 downto 0);
        component ADDNE is
          generic (N: INTEGER:=12);
```

```
          port (
          A,B    : in std_logic_vector(N-1 downto 0);
          S      :out std_logic_vector(N downto 0);
          ENABLE : in std_logic
          );
        end component;
        --configuration defaulted
        begin
        CARRY(0)(N downto 1) <= "000000000000";
        G:for I in 0 to N-1 generate
         XI:ADDNE generic map(N)
         port map (
         A,CARRY(I)(N downto 1),CARRY(I+1),B(I)
         );
         --S(I)<=CARRY(I+1)(0);
         tmp_s(I)<=CARRY(I+1)(0);
        end generate;
        --S(2*N-1 downto N)<=CARRY(N)(N downto 1);
        tmp_s(2*N-1 downto N) <= CARRY(N)(N downto 1);
        S <= tmp_s(N-1 downto 0);
end;

--================================================================================
        -- TITLE : Counter
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : cpt_win.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/21 Bassam Rhou, jr. eng.
--================================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--================================================================================
        -- Copyright (c) 2008
--================================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_unsigned.all;
```

```
        use work.all;

        entity cpt_win is
        port (
             clk         : in  std_logic;
             rst_n       : in  std_logic;
             nb_pts      : in  std_logic_vector(8 downto 0);
             en          : out std_logic
             );
        end cpt_win;

        architecture behavior_cpt_win of cpt_win is
        signal tmp_cpt : std_logic_vector(8 downto 0);
        begin
          process(clk,rst_n)
            begin
              if (rst_n = '0') then
                tmp_cpt <= "000000000";
                en     <= '0';
              elsif (clk'event and clk = '1') then
                if (tmp_cpt = nb_pts - 1) then
                  tmp_cpt <= nb_pts;
                  en     <= '1';
                elsif tmp_cpt = nb_pts then  --Condition ajouté pour que en ne soit activé que la
1ere fois
                  tmp_cpt <= nb_pts;
                  en     <= '0';
                else
                  tmp_cpt <= tmp_cpt + "000000001";
                  en     <= '0';
                end if;
              end if;
            end process;
end behavior_cpt_win;

--================================================================================
        -- TITLE : Top level
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : Top_lev.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/08/27 Bassam Rhou, jr. eng.
--================================================================================
```

```
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--============================================================================
        -- Copyright (c) 2008
--============================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;

        entity Top_lev is
        generic (N: INTEGER:=12);
        port (
          x_in    : in  std_logic_vector(N-1 downto 0);
          CLK     : in  std_logic;
          RST_N   : in  std_logic;
          g       : in  std_logic_vector(N-1 downto 0);
          dvr     : in  std_logic_vector(8 downto 0);
          o_out   : out std_logic_vector(N-1 downto 0);
          ARV1    : out std_logic_vector(2*N downto 0);
          r       : out std_logic_vector(8 downto 0);
          cmp_ARV1 : out std_logic
        );
        end Top_lev;

        architecture structural_TopLev of Top_lev is
        --Declaration des signaux
        signal out_buf   : std_logic_vector(N-1 downto 0);
        signal Dt_buf_out : std_logic_vector(2*N downto 0);
        signal out_abs   : std_logic_vector(N-1 downto 0);
        signal out_add   : std_logic_vector(2*N downto 0);
        signal out_cpt   : std_logic;
        signal out_en1   : std_logic_vector(2*N downto 0);
        signal out_en2   : std_logic_vector(N-1 downto 0);
        signal out_en3   : std_logic_vector(N-1 downto 0);
        signal out_div1  : std_logic_vector(N-1 downto 0);
        signal out_mult1  : std_logic_vector(N-1 downto 0);
        signal out_sig   : std_logic_vector(N-1 downto 0);
        signal out_compare: std_logic;
```

```
signal out_case   : std_logic_vector(N-1 downto 0);
signal out_r      : std_logic_vector(8 downto 0);
signal out_div2   : std_logic_vector(N-1 downto 0);
signal r2         : std_logic_vector(8 downto 0);
signal out_mlt_ext: std_logic_vector(2*N downto 0);


--Declaration des composants
component enable is
port (
     data_en_in       : in  std_logic_vector(N-1 downto 0);
     over_cpt         : in  std_logic;
     data_en_out      : out std_logic_vector(N-1 downto 0)
    );
end component;

component enable24_24 is
port (
     data_en24_in     : in  std_logic_vector(2*N downto 0);
     over_cpt24       : in  std_logic;
     data_en24_out    : out std_logic_vector(2*N downto 0)
    );
end component;


component abs_value is
port (
     entr         : in  std_logic_vector(N-1 downto 0);
     absol        : out std_logic_vector(N-1 downto 0)
    );
end component;

component Mult_sig is
port (
     Data_sig     : in  std_logic_vector(N-1 downto 0);
     Data         : in  std_logic_vector(N-1 downto 0);
     threshold    : out std_logic_vector(N-1 downto 0)
    );
end component;

component div_N is
port (
   ARV_in    : in  std_logic_vector(2*N downto 0);
   dvr       : in  std_logic_vector(8 downto 0);
```

```
   ARV_out   : out std_logic_vector(N-1 downto 0);
   r         : out std_logic_vector(8 downto 0)
  );
end component;

component ctrl_case is
port (
    x1       : in  std_logic_vector(N-1 downto 0);
    product  : in  std_logic_vector(N-1 downto 0);
    comp     : in std_logic;
    mod_x1   : out std_logic_vector(N-1 downto 0)
   );
end component;

component ADDER_12 is
port (
    op1      : in std_logic_vector(N-1 downto 0);
    op2      : in std_logic_vector(2*N downto 0);
    reset_n  : in std_logic;
    result   : out std_logic_vector(2*N downto 0)
   );
end component;

component comparateur is
port (
    dt_1     : in  std_logic_vector(N-1 downto 0);
    dt_2     : in  std_logic_vector(N-1 downto 0);
    sup_inf  : out std_logic
   );
end component;

component Buff is
port (
    Data_in     : in  std_logic_vector(N-1 downto 0);
    Data_add    : in std_logic_vector(2*N downto 0);
    clk_in      : in  std_logic;
    rst_n       : in  std_logic;
    Data1_out   : out std_logic_vector(N-1 downto 0);
    Data2_out   : out std_logic_vector(2*N downto 0)
   );
end component;

component multiplicateur is
port (
```

```
    A,B: in std_logic_vector(N-1 downto 0);
    S: out std_logic_vector(N-1 downto 0)
   );
end component;

component cpt_win is
port (
    clk        : in  std_logic;
    rst_n      : in  std_logic;
    nb_pts     : in  std_logic_vector(8 downto 0);
    en         : out std_logic
   );
end component;

component ctrl_init is
port (
     in_case      : in  std_logic_vector(N-1 downto 0);
     in_buf       : in  std_logic_vector(N-1 downto 0);
     in_cpt       : in  std_logic;
     final_res    : out std_logic_vector(N-1 downto 0)
     );
end component;

begin

out_mlt_ext <= "0000000000000"&out_mult1;

buf1 : Buff
port map (
  x_in,
  out_add,
  CLK,
  RST_N,
  out_buf,
  Dt_buf_out
);

v_abs : abs_value
port map (
  out_buf,
  out_abs
);

ADD : ADDER_12
```

```
port map (
  out_abs,
  Dt_buf_out,
  RST_N,
  out_add
);

en_1 : enable24_24
port map (
  out_add,
  out_cpt,
  out_en1
);

cpt : cpt_win
port map (
  CLK,
  RST_N,
  dvr,
  out_cpt
);

en_2 : enable
port map (
  out_abs,
  out_cpt,
  out_en2
);

div1: div_N
port map (
  out_en1,
  dvr,
  out_div1,
  r
);

div2: div_N
port map (
  out_mlt_ext,
  "000001010",
  out_div2,
  r2
);
```

```
mult1: multiplicateur
port map (
  out_div1,
  g,
  out_mult1
);

en_3: enable
port map (
   out_buf,
   out_cpt,
   out_en3
);

sig: Mult_sig
port map (
  out_en3,
  out_div2,
  out_sig
);

comp : comparateur
port map (
  out_en2,
  out_div2,
  out_compare
);

controle : ctrl_case
port map (
  out_en3,
  out_sig,
  out_compare,
  out_case
);

controle_initial : ctrl_init
port map (
  out_case,
  out_buf,
  out_cpt,
  o_out
);
```

```
        ARV1    <= out_en1;
        cmp_ARV1 <= out_cpt;

end structural_TopLev;

--================================================================================
        -- TITLE : Top level final
        -- DESCRIPTION : Component of the spike-clipping algorithm
        -- FILE : Top_lev.vhd
--================================================================================
        -- CREATION
        -- DATE AUTHOR PROJECT REVISION COMMENTS
        -- 2008/11/25 Bassam Rhou, jr. eng.
--================================================================================
        -- MODIFICATION HISTORY
        -- DATE AUTHOR PROJECT REVISION COMMENTS
--================================================================================
        -- Copyright (c) 2008
--================================================================================

        library ieee;
        library work;
        use ieee.std_logic_misc.all;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
        use ieee.numeric_std.all;
        use ieee.std_logic_signed.all;
        use work.all;


        entity Top_lev2 is
        generic (N: INTEGER:=12);
        port (
          x       : in  std_logic_vector(N-1 downto 0);
          CLK     : in  std_logic;
          RST_N   : in  std_logic;
          g       : in  std_logic_vector(N-1 downto 0);
          dvr     : in  std_logic_vector(8 downto 0);
          outp    : out std_logic_vector(N-1 downto 0);
          r_o     : out std_logic_vector(8 downto 0)
        );
        end Top_lev2;
```

```
architecture structural_TopLev2 of Top_lev2 is
--Declaration des signaux
signal out_ARV      : std_logic_vector(2*N downto 0);
signal out_sel      : std_logic_vector(2*N downto 0);
signal out_dt       : std_logic_vector(N-1 downto 0);
signal out_Nm       : std_logic_vector(N-1 downto 0);
signal out_m        : std_logic_vector(N-1 downto 0);
signal abs_Nm       : std_logic_vector(N-1 downto 0);
signal abs_m        : std_logic_vector(N-1 downto 0);
signal out_soust    : std_logic_vector(N-1 downto 0);
signal out_dv       : std_logic_vector(N-1 downto 0);
signal out_adder    : std_logic_vector(2*N downto 0);
signal out_mlt      : std_logic_vector(N-1 downto 0);
signal out_bf       : std_logic_vector(N-1 downto 0);
signal out_bf2      : std_logic_vector(N-1 downto 0);
signal out_bf3      : std_logic_vector(N-1 downto 0);
signal abs_bf       : std_logic_vector(N-1 downto 0);
signal cmp          : std_logic;
signal out_sg       : std_logic_vector(N-1 downto 0);
signal out_r        : std_logic_vector(8 downto 0);
signal out_r2       : std_logic_vector(8 downto 0);
signal sous_sign_s  : std_logic;
signal cmp_ARV1_sig : std_logic;
signal out_dv2      : std_logic_vector(N-1 downto 0);
signal r_o2         : std_logic_vector(8 downto 0);
signal out_mult_ext : std_logic_vector(2*N downto 0);


--Declaration des composants

component Top_lev is
port (
    x_in    : in  std_logic_vector(N-1 downto 0);
    CLK     : in  std_logic;
    RST_N   : in  std_logic;
    g       : in  std_logic_vector(N-1 downto 0);
    dvr     : in  std_logic_vector(8 downto 0);
    o_out   : out std_logic_vector(N-1 downto 0);
    ARV1    : out std_logic_vector(2*N downto 0);
    r       : out std_logic_vector(8 downto 0);
    cmp_ARV1 : out std_logic
  );
end component;
```

```
component soust_sig is
port (
     d_in     : in std_logic_vector(N-1 downto 0);
     sign_in  : in std_logic;
     d_out    : out std_logic_vector(N-1 downto 0)
     );
end component;

component abs_value is
port (
    entr       : in  std_logic_vector(N-1 downto 0);
    absol      : out std_logic_vector(N-1 downto 0)
   );
end component;

component Soustraction is
port (
     opA      : in std_logic_vector(N-1 downto 0);
     opB      : in std_logic_vector(N-1 downto 0);
     reset_n  : in std_logic;
     result   : out std_logic_vector(N-1 downto 0);
     sous_sign : out std_logic
   );
end component;

component select_ARV is
port (
    ARV1       : in std_logic_vector(2*N downto 0);
    ARV        : in std_logic_vector(2*N downto 0);
    en_sel     : in std_logic;
    RST_N      : in std_logic;
    clk        : in std_logic;
    out_select  : out std_logic_vector(2*N downto 0)
   );
end component;

component Mult_sig is
port (
    Data_sig    : in  std_logic_vector(N-1 downto 0);
    Data        : in  std_logic_vector(N-1 downto 0);
    threshold   : out std_logic_vector(N-1 downto 0)
   );
end component;
```

```
component div_N is
port (
    ARV_in    : in  std_logic_vector(2*N downto 0);
    dvr       : in  std_logic_vector(8 downto 0);
    ARV_out   : out std_logic_vector(N-1 downto 0);
    r         : out std_logic_vector(8 downto 0)
   );
end component;

component ctrl_case is
port (
    x1        : in  std_logic_vector(N-1 downto 0);
    product   : in  std_logic_vector(N-1 downto 0);
    comp      : in std_logic;
    mod_x1    : out std_logic_vector(N-1 downto 0)
   );
end component;

component ADDER1212 is
port (
     o1       : in std_logic_vector(2*N downto 0);
     o2       : in std_logic_vector(N-1 downto 0);
     add_sst  : in std_logic;
     reset_n  : in std_logic;
     result   : out std_logic_vector(2*N downto 0)
   );
end component;

component comparateur is
port (
    dt_1      : in  std_logic_vector(N-1 downto 0);
    dt_2      : in  std_logic_vector(N-1 downto 0);
    sup_inf   : out std_logic
   );
end component;

component Buff2 is
port (
     D_in        : in  std_logic_vector(N-1 downto 0);
     clk_in      : in  std_logic;
     rst_n       : in  std_logic;
     D_out       : out std_logic_vector(N-1 downto 0)
   );
end component;
```

```
component multiplicateur is
port (
     A,B   : in std_logic_vector(N-1 downto 0);
     S     : out std_logic_vector(N-1 downto 0)
    );
end component;


component mem_o is
port (
     sk         : in std_logic_vector(N-1 downto 0);
     new_sk     : in std_logic_vector(N-1 downto 0);
     remplace_en : in std_logic;
     clk        : in std_logic;
     rst_n      : in  std_logic;
     oNm        : out std_logic_vector(N-1 downto 0);
     om         : out std_logic_vector(N-1 downto 0)
    );
end component;

begin

out_mult_ext <= "0000000000000"&out_mlt;

TL : Top_lev
port map (
   x,
   CLK,
   RST_N,
   g,
   dvr,
   out_dt,
   out_ARV,
   out_r,
   cmp_ARV1_sig
);

SEL : select_ARV
port map (
   out_ARV,
   out_adder,
   cmp_ARV1_sig,
   RST_N,
```

```
   CLK,
   out_sel
);

ADD1212 : ADDER1212
port map (
   out_sel,
   out_soust,
   sous_sign_s,
   RST_N,
   out_adder
);

DV2 : div_N
port map (
   out_mult_ext,
   "000001010",
   out_dv2,
   r_o2
);

multip : multiplicateur
port map (
   out_dv,
   g,
   out_mlt
);

bf : Buff2
port map (
   x,
   CLK,
   RST_N,
   out_bf
);

bf2 : Buff2
port map (
   out_bf,
   CLK,
   RST_N,
   out_bf2
);
```

```
bf3 : Buff2
port map (
   out_bf2,
   CLK,
   RST_N,
   out_bf3
);

a1 : abs_value
port map (
   out_bf3,
   abs_bf
);

comp : comparateur
port map (
   abs_bf,
   out_dv2,
   cmp
);

msgn : Mult_sig
port map (
   out_bf,
   out_dv2,
   out_sg
);

CC : ctrl_case
port map (
   out_bf3,
   out_sg,
   cmp,
   outp
);

DV : div_N
port map (
   out_adder,
   dvr,
   out_dv,
   r_o
);
```

```
        sst : Soustraction
        port map (
           abs_Nm,
           abs_m,
           RST_N,
           out_soust,
           sous_sign_s
        );

        a2 : abs_value
        port map (
           out_Nm,
           abs_Nm
        );

        a3 : abs_value
        port map (
           out_m,
           abs_m
        );

        MMR : mem_o
        port map (
           out_dt,
           out_sg,
           cmp,
           CLK,
           RST_N,
           out_Nm,
           out_m
        );

end structural_TopLev2;

--================================================================================
        -- TITLE      :    Testbench of the global module
        -- DESCRIPTION :    This is the test bench for the SC module
        -- FILE       :    Hybride_tb.vhd
--================================================================================
        -- CREATION
        -- DATE        AUTHOR           PROJECT    REVISION
        -- 2009/10/13    Bassam RHOU        XXXXXX     v1.0
--================================================================================
        -- MODIFICATION  HISTORY
```

```
        -- DATE        AUTHOR           PROJECT    REVISION    COMMENTS
--=========================================================================

        Library ieee;
        USE ieee.std_logic_1164.all;
        USE ieee.std_logic_arith.all;
        use ieee.std_logic_signed.all;
        use std.TextIO.all;
        use ieee.std_logic_textio.all;

        entity Top_lev2_tb is
        end Top_lev2_tb;

        architecture behav_tb of Top_lev2_tb is
           component Top_lev2 port (
             x       : in  std_logic_vector(11 downto 0);
             CLK     : in  std_logic;
             RST_N   : in  std_logic;
             g       : in  std_logic_vector(11 downto 0);
             dvr     : in  std_logic_vector(8 downto 0);
             outp    : out std_logic_vector(11 downto 0);
             r_o     : out std_logic_vector(8 downto 0)
           );
           end component;

          --Width of data

          --Data for clk generation
          constant PERIODE     : time    := 50 ns; --10 ns;
          constant DEMIPERIODE : time    := 25 ns; --5  ns;
          constant DELAY_INIT  : time    := 2 * PERIODE + DEMIPERIODE - 2 ns;  --Reset period

          -- Parameter to configure J
          constant J           : integer:= 1000;   --lireJ ;  -- J parameter

          --Main signals
          signal   CLK          : std_logic:='1';
          signal   delayclk             : std_logic:='1';
          signal   RST_N         : std_logic;
          signal   x            : std_logic_vector(11 downto 0);
          signal   g            : std_logic_vector(11 downto 0);
          signal   dvr          : std_logic_vector(8 downto 0);
          signal   outp          : std_logic_vector(11 downto 0);
          signal   r_o          : std_logic_vector(8 downto 0);
```

```
    file       write_result       : text open write_mode is "destination3.txt";
    file       file_data                  : text open read_mode  is "destination2.txt";

  signal in_data_buf : std_logic_vector(11 downto 0);

  begin
   gen_Top_lev2 : component Top_lev2
    port map (
     x       => x,
     CLK     => CLK,
     RST_N    => RST_N,
     g       => g,
     dvr     => dvr,
     outp     => outp,
     r_o     => r_o
    );

    --=============================================
    -- clk generation
    --=============================================
    clk_gen: process
    begin
      CLK <= '1', '0' after DEMIPERIODE, '1' after PERIODE;
      wait for periode;
    end process clk_gen;

--=========================================================================
    -- Reset signal is generated at begin of test bench execution
 --=========================================================================
    reset_gen : process
    begin
      RST_N <= '0', '1' after DELAY_INIT;
      wait;
    end process reset_gen;

 --=========================================================================
    -- g and dvr signals are generated at begin of test bench execution
--=========================================================================
    d_in_gen : process
    begin
      g    <= "000000010010", "000000010010" after DELAY_INIT;
      dvr   <= "011001000", "011001000" after DELAY_INIT;
      wait;
```

```
 end process d_in_gen;
--=======================================
 -- Put data on inputs
 --=======================================

--Modify this process to read into one target file
put_data_in : process(clk,rst_n)
  --variable data_input  : std_logic_vector(11 downto 0);
  variable data_input  : integer;
  variable data_result : std_logic_vector(11 downto 0);
  variable L,my_line   : line;
  variable status_file : boolean;
  variable count : integer:=0;
begin
  if(RST_N = '1') then
    if CLK'EVENT and CLK = '0' then
      if   endfile ( file_data) then
        assert false
          report "end of simulation"
          severity failure;                    -- allow to terminate the simulation
      else
        readline(file_data,L);
        read(L,data_input,status_file);
        x <= conv_std_logic_vector(data_input,12);

      end if;
    end if;
  else
     x <= "000000000000";
  end if;

  count :=count + 1;

end process put_data_in;

--=======================================
 -- OUTPUT result
 --=======================================

 --Modify this process to read into one target file
 put_data_out : process
  variable outputResult :line;
 begin
  wait until CLK = '1';
```

```
            wait for 2 ns;
            if(RST_N = '1') then
              wait for 25 ns;
              write(outputResult,conv_integer(outp));
              writeline(write_result,outputResult);
            end if;
          end process put_data_out;

end behav_tb;
```